THE IMPLEMENTATION OF A LAN

by QUINTIN PETER MCGRATH
B.Sc Eng (Elec), Cape Town

Submitted to the University of Cape Town in
partial fulfilment of the requirements for
a Masters of Science Degree in Engineering

September 1987

Abstract

The subject of this thesis concerns the development of a Local Area Network (LAN) for the Department of Electrical and Electronic Engineering at the University of Cape Town.

Motivation for this project was as a result of the ever increasing demands placed on the department's micro-computer training facilities by larger student intakes. The original training system consisted of a PDP 11/23 mini-computer connected via 9600 baud asynchronous links to 11 U.C.T. built micro-computers. This network topology was limiting in three ways:

1. It was slow because of the 9600 baud links and because the PDP was doing a large proportion of the processing.

2. High-level software development tools for the PDP were too expensive and would over-load the computer. Because the micro-computers have no operating system but only an "in-house" monitor program which is not able to support any high-level language utility, all high-level software tools would have to be individually developed for this particular environment.

3. Switching was impractical. Because the PDP was the hub of the network all communication between computers had to pass through it. This switching would lead to a greater processing load on the PDP, thus further degrading its performance.

A two pronged attack was used to overcome these weaknesses: firstly, by designing a high-speed (1 Mbps) LAN to provide communications between a PDP 11/23 and up to 30 U.C.T. built micro-computers, faster intercomputer communication as well as switching and resource sharing was facilitated.

Secondly, by customizing an operating system for the micro-computers, standard high-level software development tools could be used on these computers, consequently reducing the PDP's processing load.

This report details the development of this LAN, beginning first with a study of PDP and the micro-computers, with their respective operating systems. The IEEE 802 standards, a group of standards relating to LANs, are then briefly discussed. A 1 Mbps version of the IEEE 802.4 standard for Token-passing bus networks was used in this project.

The development of the hardware and software to interface both the micro-computers and the PDP 11/23 to the network with the help of a Western Digital access controller chip, the WD2840, is described.

The next level of development was the customization of Digital Research's CP/M Operating System for use on the micro-computers. Its adaption permitted the use of any CP/M program on the micro-computers, specifically Digital's CP/NET. This software package creates a network-wide CP/M system, thereby enabling remote program storage, remote print spooling and the use of other remote peripherals.

The final section of the report details the tests performed in order to verify the component parts of the LAN.

Acknowledgements

I acknowledge my heartfelt thanks to the following:

1. My supervisors Prof. H.S. Bradlow and Mr. M. Ventura for their invaluable and untiring help.

2. The Department of Posts and Telecommunications for sponsorship during my masters studies.

3. SANS and MicroCom (Eagle Electric) for the loan of their equipment.

4. Mr. A.S.S.F. da Silva, of SAPONET, for his constructive comments.

5. My wife for her support when things went their worst, and just for being there when needed.

6. JESUS without whom I could not have achieved any of this work. To Him be any glory.

Contents

Chapter 1

Introduction

1.1 History

Since its inception in 1980, the computer network within the Department of Electrical and Electronic Engineering at the University of Cape Town has undergone major changes. In that year the department acquired a PDP 11/23, 4 terminals and a printer for undergraduate computer training. The department decided upon two areas of computer tuition:

1. Control or "Real-time" programming
2. Micro-processor assembly language programming

The first of these was provided by RTL/2, a real time language which runs under the PDP's operating system. A masters student, B.G. Sherlock, developed the interface to the control hardware and designed a software simulator. (For further details please refer to reference [1].)

The second area of programming, that of micro-processor assembly language programming, was developed largely by two students, J.A. Eva and J.G. Jack. They contributed to the design and development of a micro-computer and wrote a considerable amount of support software. Their work is described in references [2] and [3].

By the start of the 1984 academic year, which was when this project began, eleven 8085-based micro-computers had been built. Because of cost considerations, no secondary storage was provided on the micro-computers.

Instead, these computers were linked, along with 6 "dumb" terminals and a printer, to separate asynchronous ports on the PDP.

In order to reduce the computing load on the PDP, distributed processing was necessary. A full-screen editor, an 8085 assembler and an assembly code run-time debugging program had been written for the micro-computers. With the aid of these programs, users could create, assemble, debug and execute their assembly code programs on the micro-computers and use the PDP, via the network, for program storage and print spooling. Terminal emulation was also provided, enabling the micro-computers to be used as unintelligent terminals to the PDP for RTL/2 compilation and execution. The micro-computer editor was again used when writing these RTL/2 programs to ensure the least possible load on the PDP 11/23.

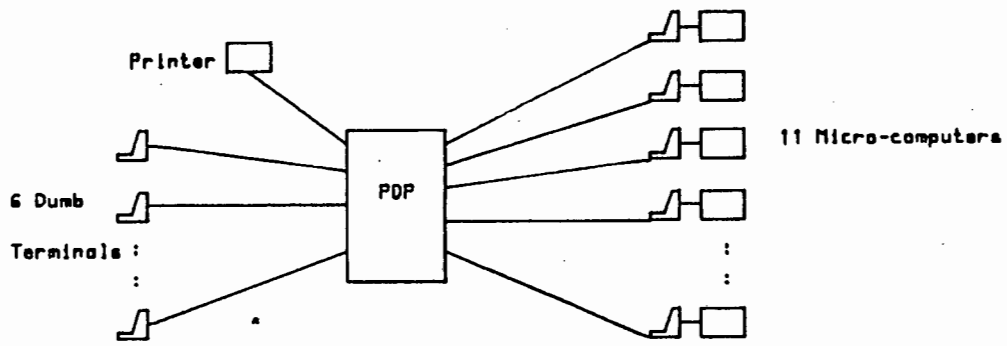


Figure 1.1 - The 1984 Network

1.2 Problems Associated with the Current Topology

Three prominent weaknesses of the 1984 network were:

1. Speed - The lines connecting the micro-computers to the PDP operated at 9600 baud. It therefore took approximately 45 seconds to transfer the longest possible file (44k bytes) between the PDP and a micro-computer (excluding the time required for mini-computer disk accesses).

2. Switching - In the star network all communications were between the PDP 11/23 and the micro-computers. For two micro-computers to communicate required the invocation of a PDP utility or some program written specially for the purpose. There are in addition a number of computers dispersed around the department which, if linked to this network, could extend the resources available to the students and in turn use the resources provided by the network. The provision of this intercommunication, however, would mean that the already overloaded PDP would have to act as a switch, thereby further reducing its performance.

3. Limited high-level software tools - Because the micro-computers only have a simple monitor program and no operating system, their high-level software tools (e.g. compilers) have to be developed within the department. Similar software for the PDP is too expensive and, if used, would again reduce the mini-computer's performance.

1.3 The Motivation

In the present computer topology, a computer (the PDP 11/23) provides services via a multiplexer unit to a number of terminals and micro-computers. Investigations revealed that with the increase in the number of tasks running on the PDP, terminal response degraded. One of the major tasks performed by the PDP in this environment is the I/O processing. This is clear if one considers that 16 terminal devices, each having its own input and output queues, have to be serviced and that the predominant task is file transfer between the PDP and micro-computers.

A suggested method of reducing this high I/O processing load is to substitute the individual point-to-point 9600 bps links between the PDP and terminal devices with a single high-speed (1 Mbps) multi-drop bus LAN. By using block transfer rather than the slow character-by-character transfer, and because only one port has to be serviced, the amount of I/O processing will be drastically reduced.

A LAN inherently provides a means of multiplexing, thus in a logical sense the original point-to-point system and the LAN system are precisely the same. That is, the PDP 11/23 provides a service to a number of terminal devices attached to it by (logical) point-to-point links. The difference is that the multiplexing function in the latter case is provided by the LAN thus off-loading the PDP.

Once it has been established that a LAN is an acceptable way of improving the teaching system as a whole, the next step was to choose the appropriate LAN architecture. It should be noted that a LAN does not simply define the transmission method and speed, and the access method, but also the protocols used for information transfer, i.e. all the layers of the ISO Reference Model (ISO/DIS 7489) are defined. The choice of the architecture would thus remove both of the above bottlenecks.

The variety of LAN architectures to choose from is almost endless nowadays because of the sudden popularity of office automation and the one-per-desk philosophy. In general, it is best to limit the choice of a LAN architecture to one that complies with an accepted standard, one of which is the IEEE 802 standards. Appendix J contains a lengthly discussion on the different IEEE architectures, their performance, reliability and error handling capabilities. It suffices here to state the conclusions that the author drew based on his discussion in Appendix J:

A version of the IEEE 802.4 Token-passing Bus Standard was chosen because:

1. It has the least complex physical level.
2. The difference in throughput and delay, i.e. the components of performance, between the Token-passing Bus and the other networks was not significant within the requirements of this network.
3. The token-passing Bus is at least as reliable as the other networks defined by the IEEE 802 standards.

Another factor that swung the scales was the availability of an access controlling IC for the token-passing Bus. This chip, Western Digital's WD2840 Token Access Controller, implements a version of the Token-passing Bus architecture which differs slightly from the IEEE 802.4 specification. Where ever possible, i.e. those sections not implemented on

the WD2840, the IEEE 802.4 standard was adhered to. It should be noted that following the IEEE 802.4 standard to the letter would have led to a complex and expensive solution.

1.4 The Proposed LAN

1.4.1 The Proposal

The aim of the project was thus to design and build a 1 Mbps Token-passing Bus LAN incorporating Western Digital's WD2840 Token Access Controller. With the aid of the services provided by the LAN, Digital Research Incorporated's CP/NET had to be used to create a distributed computer system.

From the terminal, the user will simply see a CP/M system. To make the PDP file server and printer facilities available, CP/NET was used. By running CP/NET a user may log onto a number of resource providers or "Servers", as CP/NET calls them, via the network. To allow for this the Token-passing bus architecture was used, as implemented by the WD2840. The network architecture permits the transmission of messages between any two nodes, thus allowing the possibility of more than one Server.

To improve the speed of communications, a 1 Mbps transmission speed was used. (1 Mbps was chosen as it is the maximum speed for the WD2840 access controller chip.)

The construction of this network overcomes the weaknesses of the original network in the following ways:

1. In order to use CP/NET on the network a CP/M environment had to be created on the micro-computers. This meant that the large variety of inexpensive CP/M software became available. At the same time, the processing load was moved from the PDP to the micro-computers.

2. The problem of switching is overcome because the original star network is now replaced by a multi-access bus, which inherently

allows for switching. CP/NET, in turn, provides the software for the switching.

3. Because the network speed is increased to 1 Mbps, the transmission is faster than the combination of eleven 9600 bps lines used in the original network.

This proposal is also expandable and does not require any expensive hardware - two desirable factors.

1.4.2 Introduction to LANs

According to Abrams [4], "a local network is a communications network that provides for the interconnection of a variety of data-communicating devices within a small area." In order to provide a broader view, Tanenbaum [5] lists a LAN's general characteristics as follows:

1. It will usually be small in geographical extent, of the order of a few kilometers.

2. The overall data rates are high - 1 Mbps and above.

3. Both the network and data terminal equipment (DTE) are usually owned by the same company, i.e. it is not a service provided by an external company as is the case in long-haul networks.

Fritz et al. [6] adds to the list:

1. There is usually full interconnection between all the DTEs on the network.

2. A LAN is reliable and has low data error rates - of the order of 1 in 100 million.

3. There is generally no restriction on the type of DTE used on the network.

A LAN therefore has the desired aspects of speed, interconnection and reliability. Furthermore, a LAN is essentially a switch.

1.4.3 The IEEE 802.4 Standard

When embarking on a project of this nature it would be wise to first study the relevant standards, if only to draw out a number of guidelines for development. The IEEE have prepared a suite of standards for LANs known as the 802 standards. For this particular network the IEEE 802.4 standard for Token-passing Bus networks is the relevant work. Below is the background and a brief outline of the IEEE 802.4 standard. (Should the reader require further information he is requested to refer to the ANSI/IEEE 802.4 - 1985 standard.)

With the increased demand for computer communication caused by the development of LSI technology and the subsequent creation of the microprocessor, manufacturers began designing local computer networks based on their interpretation of the ISO RM (ISO/DIS 7498). This led to a proliferation of LAN architectures. In an attempt to standardize on a certain LAN architecture the IEEE appointed a committee, known as the 802 committee.

Very early in its existence the IEEE 802 committee realized that it was unable to standardize on a single LAN architecture. Thus a number of sub-committees were formed to examine the different networks. By the beginning of 1985 standards defining four architectures had been drafted, namely:

1. ANSI/IEEE 802.3 - 1985 defining a baseband CSMA/CD bus network
2. ANSI/IEEE 802.4 - 1985 defining both a broadband and
3. a baseband Token-passing bus network

4. ANSI/IEEE 802.5 - 1985 defining a baseband Token-passing ring network

The ISO RM describes any computer communications system. Those layers that relate to the physical network configuration are the lowest two. It is these layers that the 802 standards define specifically for LANs. The Physical and Data-Link layers are redefined as the Physical (PHY), Media Access Control (MAC) and Logical Link Control (LLC) layers. The PHY layer controls the physical signalling over the network, the MAC layer controls the access to the shared medium, and the LLC layer is in charge of the logical transmission of data over the link.

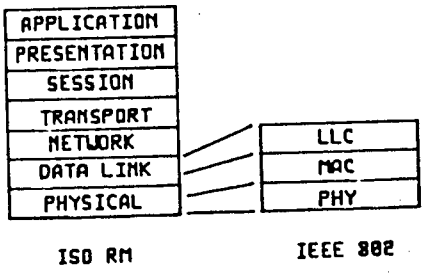


Figure 1.2 - The ISO Model and the IEEE 802 Model

1.4.4 The ANSI/IEEE 802.4 Token-passing Architecture

Token-passing is a distributed polling access method used to provide controlled sharing of a common medium by a number of terminal devices.

Essentially the Token-passing operation is this: the right to the network is passed from station to station around a logical ring on the bus network in the form of a unique control frame called the "token". The station that is addressed by the token may transmit up to a pre-defined maximum number of frames or simply pass the token to its successor. Token rotation is in descending order of station addresses, except that the lowest station passes the token to the station with the highest active address in order to complete the logical ring. Each station keeps a record of its own address, its successor's address and its predecessor's address. By fixing the rotation of the token (to either descending order, as in the

IEEE case, or ascending order which the WD2840 uses), the quick inclusion of new stations and the re-establishment of the logical ring after failure, is facilitated.

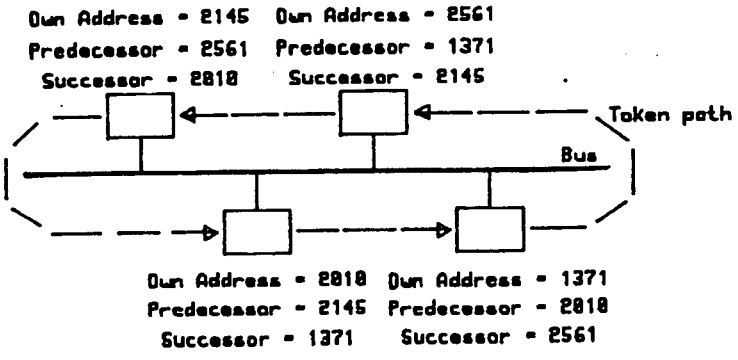


Figure 1.3 - The Logical Ring on a Token-bus Network

Although the IEEE 802.4 standard defines the lowest two layers of the ISO RM, the whole model holds true as this is a full open system, i.e. a LAN architecture defines all the layers from the Application to the Physical layer.

This method requires much maintenance. Below is a list of the minimum functions that one or more (depending on centralized or distributed control) nodes must perform:

1. Ring Initialization - at network start-up or after the logical ring has broken down, the logical ring on the network must be re-established. This requires sorting out who it is who performs the initialization, and who sends the first token.

2. Station Insertion - some method of allowing new stations onto an already active network is required.

3. Station Deletion - there must also be a method of removing an active station from the logical ring without much disruption of the service.

4. Duplicate and Lost Token - a quick and logical method must be provided for the recovery from such conditions.

The IEEE 802.4 standard defines the following method of performing the above functions:

(a) Node Addition

Node addition is achieved by the controlled contention process known as "response windows". Each node has the responsibility of searching for a new successor at regular intervals. When the node receives a token, and it is time to search for new successors, it issues a "solicit successor" frame. This invites new nodes with an address that falls between itself and the next node on the logical ring to demand entrance. The token holder will then wait for one response window, i.e. twice the network's end-to-end propagation time. At this point three events may occur:

1. No response - meaning that no new nodes are trying to join the logical ring.

2. A "set successor" frame - meaning that only one station wants to join the logical ring. The token holder will then adjust its successor value.

3. A noise - meaning that more than one station in the specified address range wants to be included in the logical ring. The node addition process now enters the second stage, contention resolution. Here the token holder sends a "resolve contention" frame and waits for four response windows. A station that responded to the "solicit successor" frame is permitted to send a second "set successor" frame, but this time it must be delayed by an integral number of response windows, based on the first two bits in the address. Should one of these stations hear another station begin transmitting in a previous window, it must no longer respond. If the token holder received a valid "set successor" frame, it continues as above in 2. If not another "resolve contention" frame is sent. Again, only those stations that responded to the previous frame may respond but this time by delaying transmission based on the next two bits of its

only mode, assuming the worst. A response is dealt with in the same way as the node addition above.

(d) Logical Ring Initialization

This process is initiated at network start-up or when one or more stations active on the network detect that the network has been inactive for longer than a predefined time. Once a timer expires, the station sends a "claim token" frame. It then listens to the network, if there is a response, then the contention must be resolved. This is done in a similar way to the node addition process. Here the claimant issues a "claim token" frame at 0, 2, 4 or 6 response windows (or slots - twice the end-to-end propagation time) after the last transmission, this time based on the inverse of the first two bits in the address. It will then listen to the network, if it hears anything it will drop the claim. If nothing is heard it tries again, this time padding the claim by an integral number of slots based on the inverse of the second pair of bits in the address. The station that transmits last assumes that it has the token. The ring is then re-built as described in the node addition section.

1.4.5 CP/NET

The popular 8085 operating system, CP/M, created by Digital Research Incorporated, was chosen for the U.C.T.-built micro-computers as they are all 8085 based systems. This proved to be a good choice as, not only is there a vast amount of high-level development software available for CP/M, but Digital Research have also produced a networking package, CP/NET, which enables resource sharing across a network.

CP/NET logically divides the networked computers into two categories, those which provide the facilities, called "Servers", and those which use the resources provided by the Servers, called "Requesters". On the U.C.T. network, the PDP would be the Server and the micro-computers, the Requesters. The micro-computer users would be able to allocate up to 16 logical disk drives, a printer and a console as external or "networked"

devices. Thus the CP/NET user creates multiple sessions or point-to-point links between his own computer (the Requester) and the one or more Servers. All networked devices are perceived as local by the user, but are accessed across the network by CP/NET.

Digital Research Incorporated supplies CP/NET in assembled code with only the user interface and the Session/Transport Layer interface defined. It is the task of the system designer to provide the lower layers of the ISO RM, i.e. the Transport and Network Layers and those layers defined by the IEEE 802.4 standard. In this network the WD2840 and the author's software and hardware provide the IEEE Layers while the Transport and Network Layers are empty because of the simplicity of this particular LAN.

1.4.6 The WD2840

The WD2840 TAC is a chip which performs the media access control functions on a multi-access bus (as described in section 1.4.4 above). This allows up to 254 computing devices to use the common bus without fear of data loss. The TAC has 16 registers which are used to control the activity of the chip. Thus to use this IC in a network requires hardware and control software to interface it to the physical network and the logical Network Layer in the computer system.

With this hardware and software the WD2840 fulfills the tasks of the three IEEE layers.

1.4.7 The System as a Whole

In the overall sense, the computer network had to look like a CP/NET system. On the micro-computers, firstly, this required the creation of a CP/M environment. To enable the use of CP/M on the micro-computers, the hardware-related portion of CP/M, its BIOS, was modified. The major adaptation was to provide a local "disk drive" on each computer. To achieve this the 128k byte EPROM card contained in the micro-computers,

previously used to store utility programs, was made to look like a CP/M (Read-only) floppy disk. The disk accessing routines in the Basic Input/Output System (BIOS) section of the CP/M operating system were adjusted. The data stored on the EPROM card was also put in the required CP/M disk format.

The CP/NET programs were then run under CP/M. To provide the IEEE layers, the WD2840 and the associated hardware and software was included.

Secondly, the PDP had to be made to look like a CP/NET Server station. In order to use CP/NET on the network, software is required on the PDP to translate the CP/NET requests received from the network into PDP operating system calls and to return the appropriate responses to the Requesters. This section is a project in itself as it requires an in-depth study of the functioning of the PDP's operating system.

An interface card and the driver software, similar to the micro-computer's software, was provided.

Diagramatically the system could be described as follows:

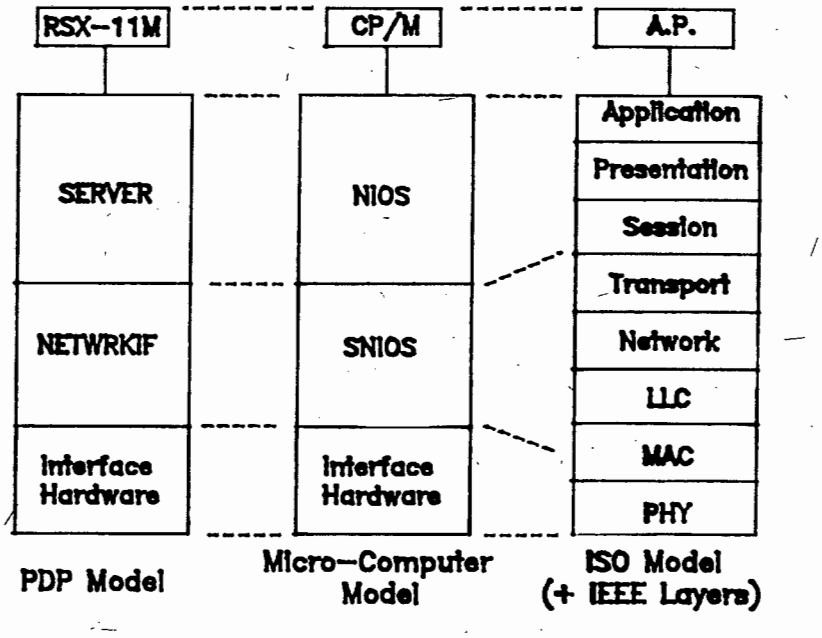


Figure 1.4 - The U.C.T. LAN and the ISO RM

1.5 Design Status

Only the part of the LAN designed by the author is complete at this stage. That part consisted of designing the hardware for both the micro-computers and the PDP, writing the software to control the hardware on the micro-computers, and adjusting the CP/M and CP/NET software for the micro-computer's hardware. A test program for the PDP hardware was also written.

Those sections still to be completed by another (unknown) student are the software interface to the PDP's executive (the "device driver"), to enable communication via the network, and the higher layers of Server software on the PDP to provide CP/NET with access to the PDP's resources.

1.6 Guide to the Rest of the Report

Chapter 2 describes the PDP 11/23 and the U.C.T. Micro-computers with their associated operating systems, RSX-11M and CP/M.

Chapter 3 describes the CP/NET software and its adjustment of CP/NET for use on the U.C.T. Micro-computers. A guide to the implementation of it on the PDP is also given here.

Chapter 4 describes the development of the network interface hardware for the micro-computers and the PDP 11/23.

Chapter 5 details the development of the network interface software for the two machines.

The final chapter, chapter 6, provides test results and draws a number of conclusions from the work performed.

The appendices that follow contain the software, the hardware circuit diagrams and a user guide for the network programs.

Chapter 2

CP/M and RSX-11M

2.1 Introduction

This chapter deals with the two operating systems, CP/M and RSX-11M. It is in these operating systems that the ISO RM's Application processes will run.

The purpose of the chapter is to give the reader an overview of the two operating systems. The customization of the CP/M Operating System for use on the U.C.T. micro-computers is also dealt with, as are three utility programs written by the author which enable effective use of CP/M on the micro-computers. Figure 2.1 below shows diagramatically the section of the project about to be discussed.

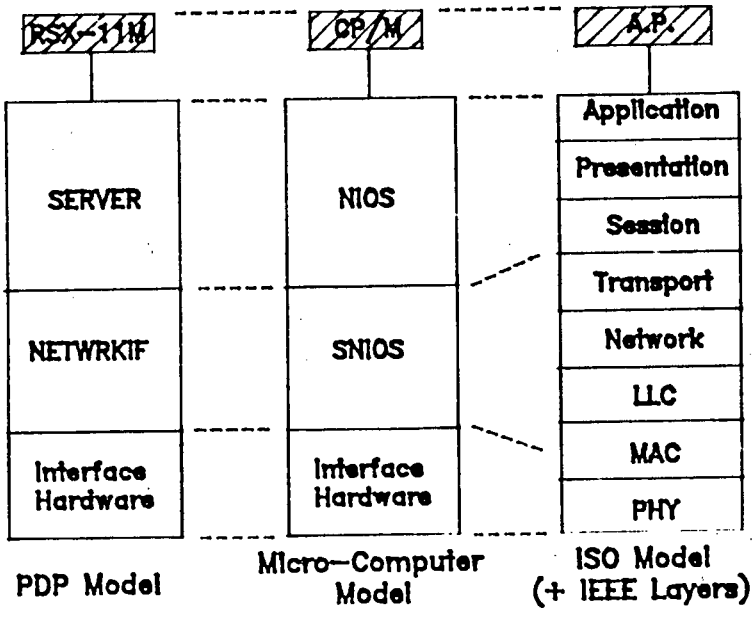


Figure 2.1 - Chapter Topic

2.2 The ISO RM's Application Process

The section of the open system model known as the application process is "an element which performs the information processing for a particular application." It can be a manual, computerized or physical process, e.g. an ATM (Automatic Teller Machine) user, an executing program, or a dedicated computer controlling some industrial process.

Iteis in the operating system evironment that the application process will run.

2.3 CP/M and the Micro-computers

2.3.1 The Micro-computers

The micro-computers were designed and built by the students and staff of the Digital Systems Division of the Department of Electrical and Electronic Engineering at U.C.T. Their design is based on the South African Standard Bus [7] (SABus) backplane. The use of this backplane allows the computer to be divided into a number of logical sections, each of which may be implemented on a separate computer card. These cards are then slotted into the SABus backplane.

Each U.C.T. micro-computer contains the following computer cards:

1. An 8085 micro-processor card.
2. A 64k byte RAM card.
3. A card holding 128k bytes of EPROM.
4. A video and keyboard controller card.
5. A communications card providing one parallel and two serial ports.

A "monitor" program, situated in an EPROM on the micro-processor card, enables the user to perform a number of simple tasks which include

examining and setting the contents of memory, adjusting port values, initiating the cpu execution from a specified point in memory and invoking utility programs stored on the EPROM card.

2.3.2 The CP/M Operating System

CP/M is a Disk Operating System (DOS) designed to run on a micro-computer system using either an 8080, 8085 or Z-80 micro-processor as its cpu. A DOS, in a general sense, enables the user to communicate with the disk drives and other peripherals, thereby facilitating reading, storing and running of programs.

CP/M is made up of a number of component programs. The core is the BDOS (Basic Disk Operating System) which controls the logical part of peripheral access. The next section, the Console Command Processor (CCP), has the task of interpreting the command line typed in at the console, recognizing a number of keywords and acting on them. The CCP also initiates the loading of user programs from disk, by means of the facilities provided by the BDOS. The third section of CP/M, BIOS (Basic Input/Output System), is different for each computer system as it is hardware dependent. This section of the software contains the code required to interface to the peripheral hardware. All three programs, the BDOS, BIOS and CCP are loaded into the top of memory. Apart from 255 bytes at the start of memory used for system parameters and temporary storage, the rest of the RAM is available for user programs.

When a program accesses a peripheral it issues a call to the BDOS entry point stating the request number and supplying the appropriate variables. The BDOS then breaks the request into its logical parts, and issues a number of calls to the BIOS. The responses are interpreted by the BDOS and returned to the calling program.

There are a large number of utilities available to the CP/M user, and for a comprehensive list the reader is requested to refer to reference [8]. The CP/M utilities which are used most frequently are:

1. DIR - for requesting a directory or catalogue of the programs on a specified disk.
2. TYPE - used to print the specified file to the console.
3. PIP - a utility which can be used to access the peripherals.

To run an executable file, the user simply types the name of the file, e.g. SORT«CR». This is interpreted by the CCP which instructs the BDOS to load the SORT file. It is then executed. On completion the CCP is again invoked to wait for further instructions.

2.4 RSX-11M and the PDP

2.4.1 The PDP 11/23

The PDP 11/23 is a mini-computer which has as its cpu a 16-bit LSI-11 micro-processor. A more complex backplane is used here, namely the Q-Bus [9]. It is more complex in that the backplane simulates a small network using a parallel bus. All transmissions across the bus are supervised by a number of control lines ensuring that data is transmitted correctly.

The U.C.T. computer comprises 128 kbytes of RAM, three disk drives and facilities for 20 asynchronous terminals.

2.4.2 The RSX-11M Operating System

The RSX-11M operating system is a multi-user and multi-tasking operating system. A user may therefore log onto the computer under a particular account, which is isolated from all other users of lower priority, and run one or more programs or utilities. This multi-user, multi-tasking ability is achieved by the time-sharing principle, i.e. each running task is given a fraction of cpu's time.

The centre of the RSX-11M operating system is the Executive. This is the kernel and is the logic behind the operating system. To interpret the user's command lines, a command line interpreter (CLI) program is used, e.g. MCR or DCL. The executive's interface to the peripherals is provided by the "device drivers", which are dedicated peripheral control programs.

Program access to a peripheral is via a call to the executive, called a QIO. This places a request in the executive's I/O queue. The request will contain such information as the peripheral addressed, the type of request (e.g. read a block of data), the timeout limit, and the internal flag to be set on completion of the request. The executive will initiate the appropriate device driver and order it to perform the task required. Once again the response is interpreted by the executive, and returned to the calling task.

When an asynchronous event is expected, e.g. the input from a terminal, the task could use an asynchronous system trap (AST), which instructs the executive to call the running task when the specified event occurs.

On a well developed system like the PDP 11/23 there are a large number of utilities available to the user. Here are a few that are relevant to the present discussion:

1. HEL - used to log onto the PDP as a user under a specific account.
2. BYE - the utility used to log off the PDP.
3. PIP - is the utility which enables limited control of the peripherals.
4. RUN - is used to begin the execution of a task.
5. SET - may be used to adjust a variety of system parameters relating to the users interface.
6. BRO - allows the user to send messages to one or more terminals.

2.5 Implementing CP/M on the Micro-computers

The RSX-11M Operating System was already running on the PDP but CP/M had to be customized for use on the micro-computers as follows:

The major problem encountered while implementing CP/M on the micro-computers was the lack of a disk drive on the computers. This problem was overcome by making use of the EPROM card, already present in the system, to provide a form of read-only disk drive. The EPROM card previously contained a number of utility programs, e.g. the assembler and the editor.

The card's hardware consists of two address latches, an EPROM counter latch and up to sixteen 8k by 8 bit EPROMs (2764s) with the appropriate bufferring. To access the data, the address and EPROM number are written to their respective latches after which data is read.

Because the micro-computers may be used as stand-alone computers, the utility programs must be available should the operating system not be loaded. To enable this the start of the "disk" on the card is uniquely marked. When the operating system is loaded the cold start loader locates the start of the "disk" and informs the BIOS routines. These routines then apply the appropriate offset when accessing the EPROM card.

The method used for program storage in the stand-alone system is different to the CP/M disk storage. The former uses two unique bytes to indicate the start of file. Following this marker is the program name and its length. The contents of the file then follow. CP/M, on the other hand, uses a directory which contains a list of the programs stored on the disk and the number and position of the sectors containing the programs. Thus to access a file each sector position must be calculated. Converting from one method to the other is clearly not practical. Figure 2.2 below shows a map of the two methods. A combination of the two methods can be used on condition that the CP/M "disk" follows the stand-alone storage.

The customized version of the BIOS may be found in appendix A of this report.

2.6 The Cold Start Loader

On station start-up there must be some way of loading the operating system into memory and initiating execution. This is the task of the Cold Start Loader program, a listing of which may be found in appendix B. It writes the three DOS component (BDOS, BIOS and CCP) programs into memory and finds the start of the "disk" on the EPROM card. The cold start program then initiates the DOS by calling the BIOS Cold Boot routine. This routine initializes the system parameters and then executes the Autoload command line (in this case CP/NET is loaded). By setting this line to contain nulls the micro-computer can be used as a stand-alone CP/M system, i.e. CP/NET will not be loaded.

2.7 The EPROM Management System

EMS, the EPROM Management System program, enables manipulation of the programs stored on the EPROM card. It is a short PASCAL-MT+ program which runs on a standard CP/M computer that has at least one disk drive. (A listing of EMS may be found in appendix C.)

On a standard CP/M system, each program stored on disk is broken up into 128 byte blocks and written to (128 byte) sectors. The position of these sectors is recorded in the directory, along with the file name and a count of the number of sectors used. The directory entries are stored sequentially in the first sector of the disk.

To add a new file to the EPROM disk, EMS reads the specified file from disk into RAM, breaks it up into the required 128 byte blocks and then writes it to a number of files on disk. These files, each a maximum of 8 kbytes long, define the data that must be contained in each EPROM of the

EPROM card. The entry is then generated and added to the directory contained in the "directory" EPROM file.

To delete a program, EMS simply adjusts the directory EPROM file to exclude the specified file. This is done by moving all the entries that follow the deleted file up by one entry position.

The data contained in these EPROM files generated by EMS may be used to program the EPROMs which are then inserted into the EPROM card.

Chapter 3

CP/NET

3.1 Introduction

This chapter provides an overview of CP/NET and its working. This package consists of two sections, the commercially supplied software, NDOS, which is the interface to the CP/M software, and SNIOS, the software that provides the link between NDOS and the network. This chapter is mainly concerned with NDOS, although CP/NET as a whole is also dealt with. Because NDOS is not adjustable, it was thus used as is on the micro-computers. The PDP uses a different operating system, so the PDP version of NDOS has to be written from scratch. This program, called the SERVER, is a task running on the PDP which will provide the user access to the network. It will call the operating system executive to initiate NETWRKIF, the LAN device driver, to interact over the network, and use inter-task communication to converse with the user/application task. Guidelines are provided herein to aid the student who is to embark on this project.

The sections of the project being discussed in this chapter are:

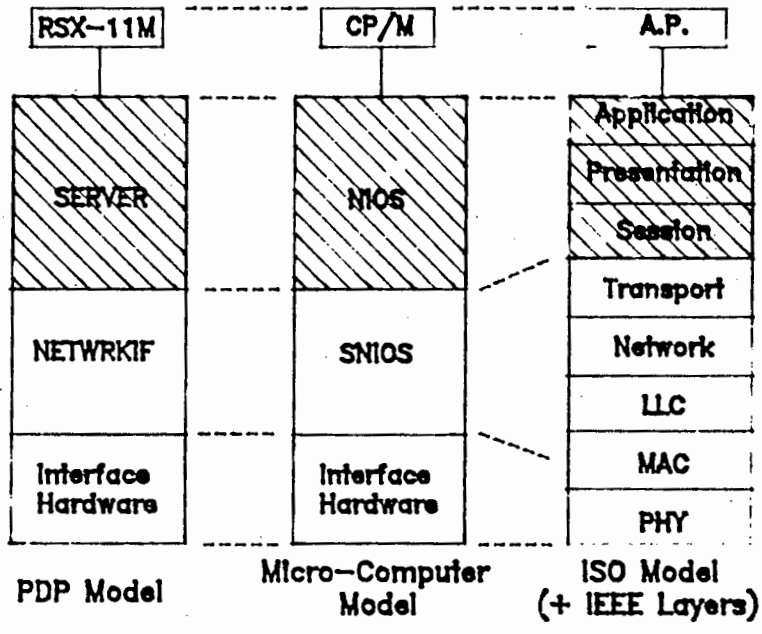


Figure 3.1 - Chapter Topic

3.2 ISO Layers

The three layers being considered in this chapter are those that provide the Application Process (AP) with an interface to the network. The lower layers are concerned rather, with the actual transmission of the information across the network.

Briefly, the Application Layer is the AP's window to the network, and so provides a means of communicating with other APs attached to the network. It has to transform the data into a format that can be sent over the network. In some networks where distributed processing is performed, the Application Layer has the task of controlling the resource usage. The AP/Application Layer interface simply defines the transmission and reception of data.

The Presentation Layer is responsible for insuring that the format of the information is acceptable to the Application Layer, and thus the AP. Some examples of Presentation Layer functions are syntax changes (e.g. the conversion between ASCII and EBCDIC), encryption and decryption (to provide secrecy) and data formatting (e.g. space and character compression). The interface to the Application Layer defines the transmission of Application Data Units as well as the parameters to select the required presentation functions.

The Session Layer manages the connection between two presentation entities in that it is in charge of the connection and disconnection of the link (or "session"). This requires the transformation of an AP address into a Session Layer address. It is also in control of the dialogue, i.e. whose turn it is to transmit next and the speed at which data is presented for transmission. The Presentation/Session Layer interface defines the transmission of Presentation Data Units, synchronization control and parameters for exception reporting.

The NDOS and SERVER programs perform the services required by these three layers. Note that some of the ISO RM's services are not performed as they are not required in this environment.

3.3 The CP/NET Package

3.3.1 Component Programs

When a micro-computer takes part in a network it loads the CP/NET programs, namely, NDOS and SNIOS. The NDOS (Networked Disk Operating System) is the user interface part of the package, i.e. it performs the Application, Presentation and Session Layer functions and provides their services. The SNIOS (Slave Network Input/Output System) performs the network oriented services, i.e. the rest of the ISO RM services. SNIOS is therefore entirely network dependent while NDOS is network independent.

In order to achieve network access the NDOS intercepts all peripheral calls to establish whether they are for local or networked devices. If a call is for a local device it is passed to the BDOS and the standard CP/M access continues, i.e. the BIOS is instructed to perform the required function, say, read from disk. Responses are returned to and interpreted by the BDOS, and the results are returned to the calling program. If, on the other hand, the peripheral call is for a networked device, the NDOS creates a packet and sends it across the network to the Server station using the network interface provided by the SNIOS program.

On the Server side, the packet received from the Requester is interpreted and the appropriate operating system calls are issued. When a response is received, it is relayed to the Requester via the network. This whole task is achieved by two programs, the network interface program (NETWRKIF), which performs the same functions as the Requester's SNIOS, and an operating system interface program (SERVER).

Because a number of Requesters may use the same Server and each of them requires protection from other users, a multi-user operating system must be used. On the standard CP/NET network the Server computers will run another Digital Research product, MP/M-II, a multi-user version of CP/M. On the U.C.T. network, the PDP 11/23 will be the only Server at this stage, and so the tasks normally performed by the MP/M-II Server will be performed

by a software package (still to be completed by another student) running under RSX-11M, the PDP's operating system.

3.3.2 CP/NET Utilities

The features of CP/NET fall into three categories:

1. Networked peripherals
2. Inter-computer message transmission
3. Electronic Mailing

The extra facilities available to the CP/NET user are initiated by typing the appropriate keyword. These are recognised by the extended console command processor program, CCP, and are:

1. CP/NET Loader - This is the utility used to load the CP/NET programs and to relocate and extend the CP/M Operating System programs.

2. Logon - enables the CP/NET user to attach the CP/NET computer to a Server computer as a Requester. A Server may have as many as 16 Requesters logged on at the same time. The Requesters may similarly, be logged onto up to 16 Servers.

3. Logoff - detaches the Requester from a specified Server.

4. Network and Local - allows the user to assign a peripheral to an attached Server, and to re-assign them as local. 16 disk drives, a printer and a console may be assigned as networked devices.

5. Control-P and Endlist - will begin and end an echo of the console display to the printer which may be local or remote.

6. Disk Reset - allows the user to reset the parameters of a remote disk. This is required when a disk is changed on an MP/M computer, and has

the same effect as a Control-C on a CP/M computer, i.e. it reads the directory track to ensure that subsequent access to the new disk is correct.

7. Status - will display a table of peripherals indicating which are local and which are networked.

8. Mail - initiates the menu-driven Electronic Mailing utility.

Under CP/M, the user may access BDOS directly via an assembly code program. For example, the user may reset the computer system, write characters to the console and so on. NDOS may be accessed in the same way. This is facilitated by defining 14 "function codes" which are appended to the standard CP/M function codes.

To access the network in this way, which may be thought of as a second AP/Application Layer interface, the user sets up the parameters as shown in table 3.1 and calls the NDOS entry point at 0005H. The NDOS program will perform the function and return to the calling process. (The table below gives a list of the functions available; should more information be required please refer to reference [9].)

Code (C value)	Function Name	Input Parameters	Output Results
38	Access Drive	DE: Drive vector	none
39	Free Drive	DE: Drive vector	none
42	Lock Record	DE: FCB address	A: Error code

43	Unlock Record	DE: FCB address	A: Error code
45	Set BDOS Error Mode	E: Error Mode	none
64	Login	DE: Addr of Login Msg	A: Error code
65	Logoff	E: Server ID	none
66	Send Msg	DE: Msg Address	A: Error code
67	Receive MSG	DE: Buffer Address	A: Error code
68	Get Ntwrk Status	none	A: Status byte
69	Config Table Address	none	HL: Table Address
70	Set Compat. Attributes	E: Attrib.	none
71	Get Server Config.	E: Server ID	HL: Table Address
106	Set Default Password	DE: Password Address	none

Table 3.1 - NDOS Function

At the other end of NDOS, the information is very similar except that the source and destination addresses (each 8 bits long) have been added, as well as a format control code, fixing the lengths of the fields of the buffer, and a size field, indicating the number of Session data units are in the buffer. The message buffer format is (each field is one octet long):

1. FMT - the format control field, containing a 0 for the request message and a 1 for a response from the Server.

2. DID - the destination ID field. It may have a value of 1 to 255.

3. SID - the source ID field, i.e. the address of this station. Again, the value range is 1 to 255.

4. FNC - is the function code and will be one of the C values in table 3.1, or any of the CP/M codes applicable to peripherals.

5. SIZ - gives a count of the number of message octets in the buffer. It has a range of 0 to 255, and is one less than the number of MSG octets.

6. MSG0 to MSGn - are the message octets, where n is SIZ plus one. The values of these octets is 0 to 255.

The DID and the FNC together define the particular application being addressed on the server.

It is this message buffer that must be sent across the network by SNIOS and the responses from the Server must be returned to the NDOS program in the same format.

Relating the NDOS functions to the services of the top 3 layers of the ISO RM now:

The user interface to the network (or the application layer) is provided via the function calls (specified in table 3.1) or via the utility programs.

The NDOS and SERVER programs convert the function calls into a message containing the relevant address, length and format fields.

The session is established and disconnected by the LOGIN and LOGOFF commands received either via function calls or by the utility program. The NDOS/SERVER interaction works on a strict command-then-response routine thereby defining the sequence of transmissions.

3.4 Guidelines for the PDP's Server (Part 1)

There are two distinct parts to the PDP's network package, the SERVER program which will be the interface to the PDP's operating system, and the device driver program, previously called NETWRKIF. In this section only the guidelines for the SERVER program are given, once the WD2840 control

software has been discussed, the NETWRKIF program guidelines will be given (see Chapter 5).

In the standard SERVER program in the MP/M II Server station, each Requester that logs on is given its own SERVER process. Thus as the Requester logs in the SERVER generates a new task to deal solely with the new Requester's requests. The limit of 16 Requesters at one time will define the upper limit of the memory requirement. It is suggested that the PDP's SERVER process follows the same sequence.

3.4.1 Virtual Terminals

The first task is convince the RSX-11M's executive to allow a multiple number of users to be logged onto the same physical peripheral simultaneously. There will be only one physical input from the network to the PDP. One suggestion is to use the "virtual terminal" facility provided by RSX-11M. This allows the user to generate a number of logical terminals linked to a task with the real terminal as the port for input and output. Thus there will be a multiplexing/demultiplexing task which is the first to get the message from the device driver. It will then select the correct SERVER process and pass the request on. In this way each Requester could log onto the PDP via its own virtual terminal.

The creation of a virtual terminal is a simple matter via the executive's Create Virtual Terminal function. It is possible to initiate execution of a process from a second task by means of the Spawn function. Using the Spawn function in the multiplexer/demultiplexer task would enable it to start a new SERVER process for each Requester. At the termination of the session, i.e. when the Requester Logs off the Server, the parent task could be informed and the virtual terminal deleted.

3.4.2 Drive Allocations

CP/NET allows the allocation of 16 disk drives on one or more Servers. Generally, when a user signs onto the PDP from an unintelligent terminal he is assigned a number of defaults, namely an account number, a disk drive and a terminal number. From within RSX-11M a user is able to change these defaults, so is able to access a drive other than his default drive. It should thus not be difficult to allow the user to define a particular drive (i.e. A: through P:) on the CP/NET computer as a networked on the PDP (i.e. DL0: through DL2:) and an account number, or as the magnetic tape driver, MN0:. For example, B: may be networked as DL2:[1,20], i.e disk drive 2 and account number 1,20.

A problem is that CP/NET simply uses a vector 16 bits long to indicate which drive on the Server is being selected. Because the NDOS program is unalterable (or at least without much effort), a selection of defaults would have to be set up before networking a device so that a single bit in the vector would have meaning. The vector would therefore have to either be predefined on the Server, or be set up by a program run when the user logs on. Because most of the students simply use their default account and drive, a predefined vector may well be the best solution, e.g.:

Drive A: on the server (bit 1 in the vector) is the default account and drive.
B: (bit 2) is the default account and drive 0.
C: (bit 3) is the default account and drive 1.
D: (bit 4) is the default account and drive 2.
E: (bit 5) is the default account and the tape drive.
and so on ....

3.4.3 Disk Accesses

A CP/M disk uses a 128 octet block structure, and so all access are in 128 octet blocks. This is allowed by RSX-11M although it may be more

efficient to access larger blocks. This can be done assuming that subsequent blocks are going to be requested.

More complex is the directory structure and the file control information. It would be unwise to duplicate RSX-11M's directory information and so some translation is preferable.

The file control abilities under RSX-11M are considerably more powerful than CP/M's and so the required file control facilities will easily be supplied, although RSX-11M's abilities in this field will not be used to the full effect. The required facilities include record locking (to allow access to only one user), allowing two tasks to update the same file and file appending.

3.4.4 Status and Configurations

There are a number of configuration and status tables that must be supplied by the Server on request. Most of these tables bear no relation to the PDP and so will have to be generated in the SERVER process. These tables include the configuration of the Server, i.e. which drives (A: through P:) are being used by which Requester, and the status of the network, which would have to be calculated by the SERVER process on the basis of information supplied by the NETWRKIF process.

Chapter 4

The LAN Interface Hardware

4.1 Introduction

This chapter is the first of two chapters dealing with the network interface layers of the LAN. This chapter jumps to the PHY and MAC layers of the architecture, while the following chapter deals with the LLC and Network layers. In this chapter the WD2840 Token Access Controller (TAC) and the hardware required to drive it are described, i.e. the hardware that forms the bridge between the physical network and the computer's backplane. The next chapter will detail the software required to interface it to the CP/NET programs. In this way all the network interface programs of the ISO RM (the Transport and Network Layers) and the IEEE layers which replace the Data Link and Physical Layers, i.e. the LLC, MAC and PHY Layers, will be implemented.

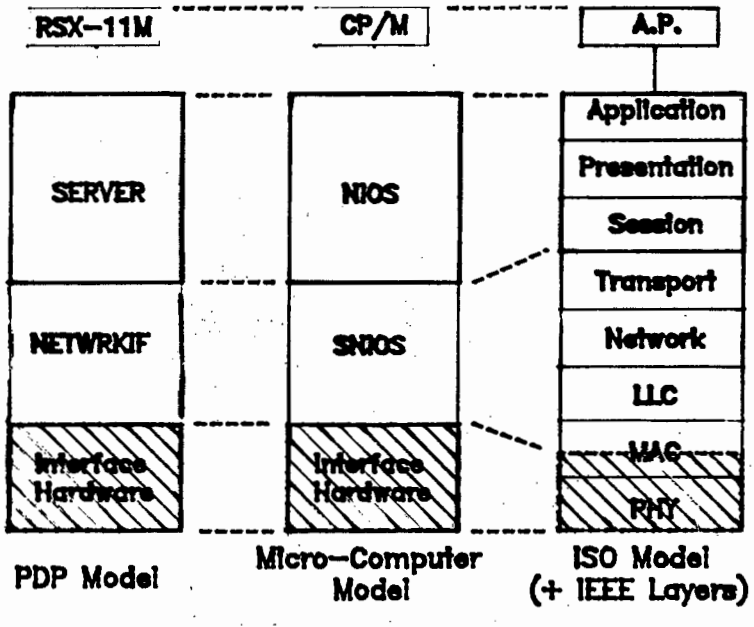


Figure 4.1 - Chapter Topic

The first section of the chapter provides the information about the micro-computer and PDP backplanes. The WD2840 from a hardware point of view is then described. Finally, the interface hardware for both the micro-computers and the PDP is described.

4.2 The ISO RM Network Related Layers

There are four layers that make up the network related layers in the ISO RM, the Transport, Network, Data Link and Physical Layers. In the LAN environment the Data Link and Physical Layers are replaced by the three IEEE layers, the Logical Link Control (LLC), the Media Access Control (MAC) and Physical (PHY) Layers.

The Transport Layer is the end-to-end network control layer and so supplies the Session Layer (and thus the higher layers and AP) with an error-free link. It will generally break the message received from the Session Layer into small blocks, and concatenate the parts of the Session data unit received from the Network Layer to make one Session data unit, or perhaps vice versa. This calls for the ability to sequence the blocks of data so that the information received is the same as the sent data. It is responsible for the end-to-end flow control and error recovery. In larger networks it may use a number of Network Layer links between the same two Transport end points, these must be controlled and addressed properly.

The Network Layer is responsible for routing the data through the network, which, in large networks, requires a number of intermediate point-to-point links. The short point-to-point links must be flow controlled and checked for errors by the Network Layer. It must be capable of resetting the network links.

The LLC Layer is responsible for the link control functions of the Data Link Layer, i.e. the delimitation and synchronization of the transmission, the sequence control over the link and any error recovery that is required once an error is detected.

The MAC Layer has the task of controlling the access to the common medium. It must thus provide access facilities, address recognition, error detection facilities and data transmission.

The PHY layer is responsible for the bit-by-bit transmission and reception of data. It must perform unit framing and frequency or amplitude conversions, e.g. Manchester Encoding/Decoding and FSK Transmission.

4.3 The U.C.T. Micro-computer's Backplane

The U.C.T. Micro-computers use the SABus [7] backplane. This backplane consists of 64 lines. Below is a description of those relevant to this project.

The first group is the 8 bi-directional data lines. The drivers for these lines must be tri-stated (i.e. in high-impedance state) when they are not driving the lines. There are 16 address lines which are again bi-directional. This is the second group of lines which, once again, must be tri-stated when not in use.

The third group is made up of a variety of control lines, those relevant to the discussion are:

1. Memory Read (MR) and Memory Write (MW), which are used by the controlling device (the cpu or a DMA controller) to activate the micro-computer's memory.

2. Output Write (OW) and Input Read (IR) are two lines used by the controlling device to receive input from and give output to an I/O device.

3. The WAIT line is an open collector driven line which is used by the addressed device to inform the controlling device that it is not yet ready for the data. This line is most commonly used by the RAM to

indicate to the cpu that it is not ready for data or that the data on the backplane is not valid.

4. HOLD and Hold Acknowledge (HOLDA) are two lines connected to the cpu which enable the use of more than one controller on the backplane, e.g. a DMA controller will activate the HOLD line to indicate that it has information to pass to the RAM. Once the cpu has completed its present cycle, it relinquishes the backplane and indicates to the DMA controller via the HOLDA line. The DMA controller performs its transfer and de-activates the HOLD line allowing the cpu to continue as before.

5. RESET is a line, activated by the cpu, which initiates the resetting of all the cards attached to the backplane.

All the levels on the SABus backplane are TTL levels. The different timing diagrams for the relevant read and write cycles may be found in Appendix D.

4.4 The PDP's Backplane

The PDP's Q-Bus [8] backplane is similar to the SABus although it has much stricter specifications for the attached devices and their voltage levels because it is run at a higher speed. All devices attached to the backplane must use open collector drivers, with high-impedance input buffers. Below is a description of the relevant lines.

The 16 address/data lines (BDAL0 - BDAL15) are multiplexed under control of the synchronization line (BSYNC) and the two input/output control lines BDIN and BDOUT. The write cycle line (BWTBT) is also used during the addressing phase to indicate if a write or read cycle is to follow and whether it is a byte or word long. The addressed device indicates acceptance of the data (for a write) or the stability of the data (for a read cycle) via the reply line (BRPLY).

In the Q-Bus, 128 kbytes of memory may be addressed, but the top 8 kbytes are set aside for the I/O page. Thus to reduce the decoding in I/O devices a control line (BBS7) is activated each time the address is in the I/O page.

The PDP's interrupt structure is designed to enable both priority levels and, within each priority level, position dependence. The high priority devices are serviced first. If there are a number of devices with the same priority interrupting simultaneously, they are serviced in the order of their proximity to the cpu. This structure is illustrated below in Figure 4.2.

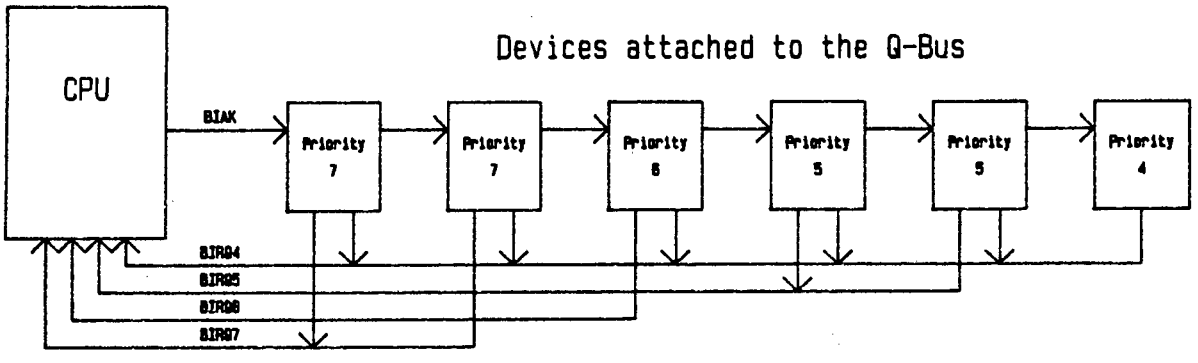


Figure 4.2 - PDP Interrupt Structure

Under interrupt conditions the device will activate the interrupt lines (BIRQ4 to BIRQ7) which gives the priority of the interrupt. Once a response is received from the cpu, via the BIAKI line, the vector is given on the BDAL lines under control of the BDIN and BRPLY lines.

To initialize the devices attached to the backplane the cpu activates the BINIT line.

The timing diagrams for the read, write and interrupt cycles may be found in appendix G.

4.5 The WD2840 Hardware Interfaces

This section describes the three interfaces on the WD2840 TAC chip. These interfaces are a DMA interface, a network interface and an interface to the 16 host-accessible status/control registers on the TAC. The WD2840 timing diagrams may be found in appendix D.

4.5.1 The DMA Interface

The DMA interface permits the transfer of frames and frame related counters between the WD2840 and RAM. The frame related counters give such information as the number of frames received, the number of transmission retries and the number of erroneous frames received.

This interface consists of 16 address lines (A0-15) and 8 data lines (DAL0-7). The TAC provides two DMA request lines (DRQ0 and DRQ1) and a DMA request granted line (DACK) which are used to control the DMA access and provide the RAM control lines during DMA.

4.5.2 The Network Interface

The network interface provides the signals required for a Non-Return to Zero Inverted (NRZI) modem interface. The WD2840 uses synchronous transmission and so provides lines for both the receive and transmit clocks (RC and TC) and their respective data lines (RD and TD). The other network interface lines are request-to-send (RTS), clear-to-send (CTS) and signal quality (SQ).

4.5.3 The Control Interface

The WD2840's activity is controlled by 16 host-accessible status/control registers which may be accessed via the third interface. These registers provide such status information as the reason for the

interrupt, whether the previous command is complete and present successor's address. The host computer, on the other hand, will provide commands like attempt to enter the logical ring, accept new successor address and transmitter or receiver enabled.

This interface consists of four address lines (IA0-3), eight data lines (DAL0-7) and the three control lines, being the chip select (CS), read enable (RE) and write enable (WE).

4.6 Design Philosophy

The design of both of the NIU cards (and the control software described in the next chapter) employed Julius Ceasar's philosophy of "divide and conquer". This was achieved by breaking down the task into a number of logical blocks which were developed and debugged as separate entities. Further, their development followed the steps given by Short [10] for the design of a micro-processor based system, shown diagramatically in figure 4.3 below.

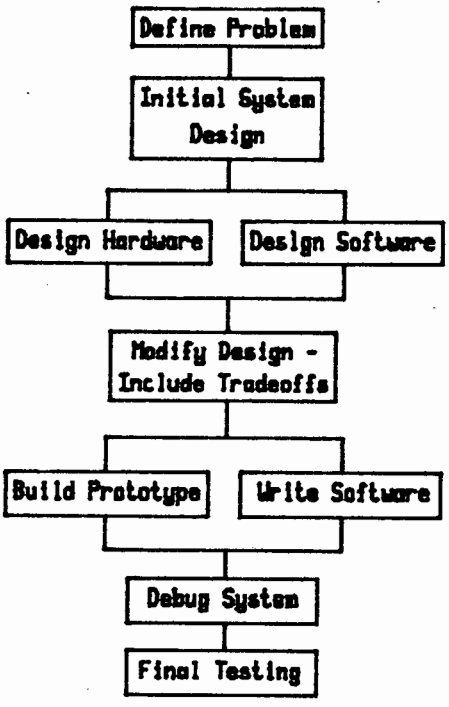


Figure 4.3 - Steps in designing a Micro-processor based system

From the references [11,12,13], it is evident that a stand-alone NIU is a popular implementation. This method was not employed here as it would create a separate unit that would require its own power supply, making it both costly and bulky. Secondly, because of the 1 Mbps operating speed, an additional high speed interface to the computers would have to be designed.

Instead, two compact computer cards were developed, one to slot into the micro-computer's backplane and the other into the PDP's Q-bus.

These circuits were initially designed using logic gates and prototyped on wire-wrap board to verify their functioning. Once fully tested, a new version, incorporating the logic into Programmed Array Logic chips (PALs), was designed and tested for each NIU. The PALed designs were then transferred to printed circuit boards.

PALs were used in the final design instead of logic gates. This was for three reasons: their speed, their cost and their size. With regard to speed, the PAL is able to realize even complex functions within 20 nanosec, whereas the discrete solution's relation to time is determined by the number of gates used.

Secondly, on average a PAL replaces between 4 and 6 logic gate ICs but only costs in the order of 3 times that of a single logic gate chip.

One PAL is also much smaller than 5 or 6 logic gates, thus space is saved on the printed circuit board. A certain degree of flexibility with regard to the input and output pins on the PALs also allows for more efficient printed circuit board layouts.

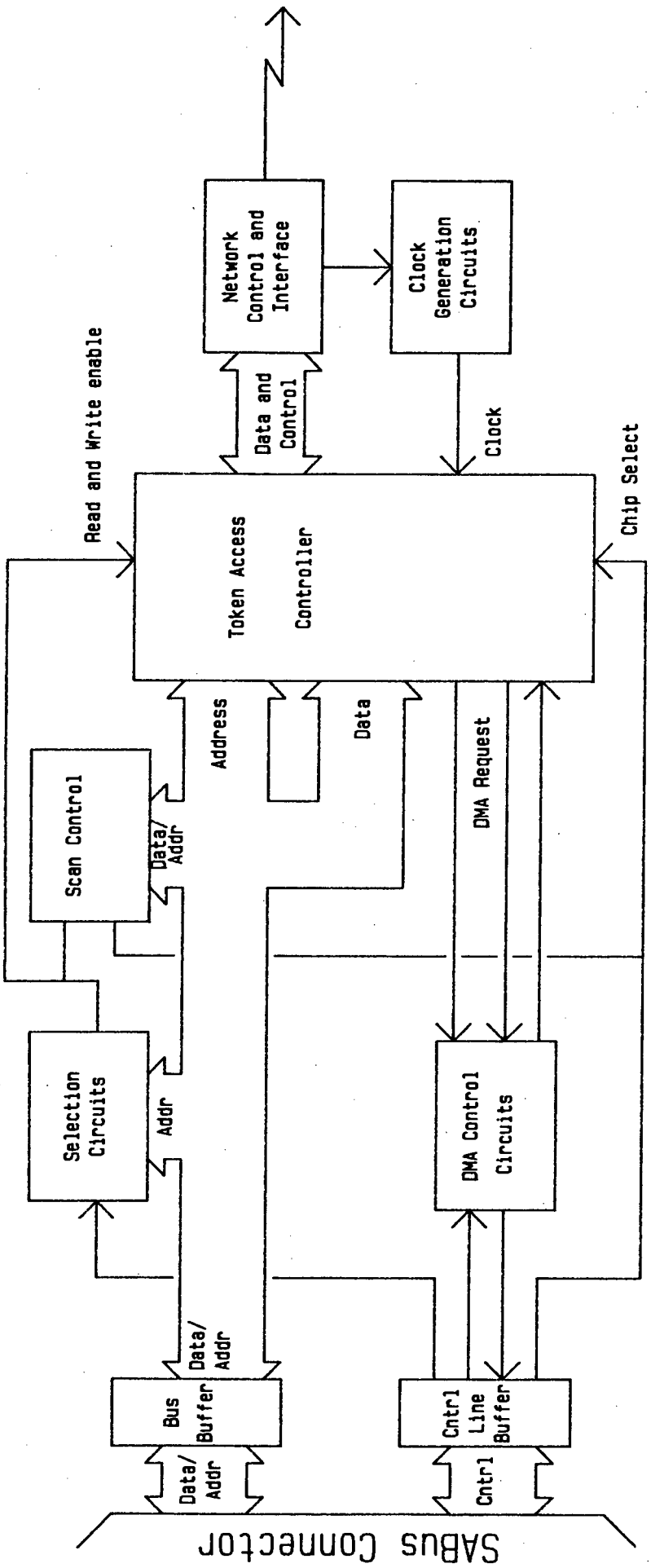
4.7 The Micro-computer LAN Card

4.7.1 Design Description

Below, in figure 4.4, is a block diagram of the Micro-computer interface card.

(a) The DMA Interface

The WD2840 is a real-time device which requires immediate access to the RAM where the data frames and control information is stored. One possible method of ensuring this access is to provide it with a block of local RAM which could be accessed at will. This RAM would be separate from the host's RAM and so would not be affected by host activity. The computer's cpu would only have access to this memory when the WD2840 was not using it. A second possible method is to set aside a part of the host's RAM for use by the WD2840. The WD2840 DMA control facilities could then be used to control RAM access.


Network Control and Interface
Clock Generation Circuits
Data and Control
Clock
Read and Write enable
Chip Select
Token Access Controller
Address
Data
DMA Request
Scan Control
Data/ Addr
Selection Circuits
Addr
DMA Control Circuits
Data/ Addr
Bus Buffer
Cntrl Line Buffer
Data/ Addr
Cntrl
SABus Connector

The first method above permits the use of a number of real-time devices, such as disk drives and asynchronous terminals, because the NIU can continue to access its local memory without holding up the rest of the computer's activity. When the NIU requires service, e.g. frame reception complete, it can flag the cpu which will service the interrupt when it has time.

The PDP's NIU used this method because of the other real-time devices attached to the computer which could not be stopped at will. In the case of the micro-computers, on the other hand, no other real-time devices are attached to the computer and so part of the host's RAM was set aside for NIU use. This also allowed for a cheaper solution.

In the micro-computer, bus (and therefore RAM) arbitration is achieved by activating the SABus HOLD line under control of the WD2840 DMA request lines. This bus line is connected to the 8085's /HOLD input, which when asserted, instructs the 8085 to complete its present cycle and then relinquish the bus. Once the bus is available the WD2840 is informed via the SABus hold acknowledge (/HOLDA) line. The WD2840 then uses its DMA request lines to perform memory reads and writes by activating the SABus Memory Read (/MR) and Memory Write (/MW) control lines. Dynamic RAM is used in the micro-computers and so the NIU has to accept short delays whenever the RAM requires refreshing.

The TAC address and data lines are buffered to provide isolation and prevent overdriving the WD2840's outputs.

(b) The Control Interface

The second section of the WD2840's host computer interface allows the host cpu to control the functioning of the TAC. This is achieved by adjusting the values of 16 status/control registers on the WD2840. There are two limitations on the access of these registers - they may not be accessed while the TAC is performing a DMA transfer and there is a minimum

inter-access time that must be upheld. The first of these limitations cannot occur in this implementation as the cpu is in hold state.

Should the cpu attempt to access the TAC during the inter-access time, the SABus WAIT line is activated until the TAC is ready. The assertion of the WAIT line causes the cpu to enter an idle state.

Access to the TAC registers is gained by activating the Read Enable (/RE), Write Enable (/WE) and Chip Select (/CS) lines on the WD2840. The /RE and /WE pins are connected to the Input Read (/IR) and Output Write (/OW) lines on the SABus backplane. The four most significant lines of the 8 address lines are used to select the card and thus the TAC. The position of this card in the 8085's I/O page is chosen by selecting a 16 byte boundary address via mini-jumpers on the interface card. The four least significant address lines are used to select one of the 16 TAC registers. The data lines are buffered through bi-directional, tri-stateable buffers controlled by a combination of the /IR, /OW and the card select lines.

(c) The Network Interface

The WD2840 has the necessary signals to interface it to a NRZI modem. The modem has to provide both the receive and transmit clocks, the required signal levels for the network as well as network isolation.

The IEEE 802.4 standard requires Manchester encoding when Phase-continuous FSK is used. Since this PHY layer may be used on the U.C.T. LAN in the future (once the Signetics FSK chip pair becomes available), manchester encoding was included. This method also provides a quick and reliable method of clock recovery from the transmitted data.

The encoding is performed in this LAN by a Harris Digital IC, the HD-6409. This chip provides a fixed frequency Transmit clock and derives a Receive clock from the received data. By connecting this chip to the TAC, data may be clocked into the HD-6409 (from the TAC's TD pin) on the rising edge of its Encode Clock (ECLK). The TAC uses the inverted image of the

encoder's Decode Clock (DCLK) to synchronize to the data stream being received (on its RD pin) from the HD-6409.

The Manchester Encoder also provides an output to indicate whether the data being received is valid. This output is connected via an inverter to the Signal Quality (/SQ) input of the WD2840. If, at any stage during the frame reception, the encoder discovers an invalid Manchester character, the Signal Quality line is deactivated and the TAC discards the frame.

The encoder's 16 MHz clock input is supplied by a passive crystal oscillator circuit. As an output from the encoder there is a 16 MHz TTL square wave which is used to drive both a divide-by-8 counter for the WD2840's clock input, and the timers for the synchronous circuitry.

On the network side of the HD-6409, an RS-422 receiver/driver IC drives the twisted-pair cable directly. This chip is enabled by an inverted image of the TAC's RTS line. Its task is to provide the correct signal levels and network isolation as defined by the EIA RS-422 standard.

(d) The Solicit Timer

The TAC can use either centralized or distributed solicit control. The vulnerability introduced by a centralized controller led to the choice of distributed control for this implementation. This choice also complied with the IEEE 802.4 standard.

The solicit timer, which is part of the MAC layer, defines when the station must search for a new successor. As there is no real-time clock on the computer, which would generally be used to define the inter-solicit period, a solicit timer had to be included in the hardware. This timer was not implemented in software due to the variety of programs that will be run.

At regular intervals (determined by this hardware timer) the TAC is instructed to scan until a (new) successor is found. This is achieved in

hardware as follows: The value of the CR1 register (which controls the successor address value) is read into an octal flip-flop. The required bit is set and the new value written back to the TAC's CR1 register. The TAC will then set its successor's address to the value given, which is pre-set to its own address plus one at start-up. The timing is provided by a PAL clocked by the 16 MHz signal from the Manchester Encoder. To address the CR1 register, a buffer which has its inputs hardwired to the correct value, is selected. Should the host cpu attempt to read the registers during this period, the SABus WAIT control line will be asserted until the end of the cycle.

(e) Reset Circuitry

The reset line from SABus backplane is used to reset both the TAC and the Manchester Encoder. A 10 msec reset pulse is supplied by the cpu via the backplane, thus no extra circuitry is required apart from the usual buffering as specified by the SABus Standard.

A full circuit diagram of the micro-computer NIU, the PAL equations and timing diagrams are given in appendix D.

4.8 The PDP Interface Card

4.8.1 Problem Description

The PDP's NIU must be designed to work with a number of other real-time devices, namely 3 disk drives, a tape drive and asynchronous terminals. The tape drive, disk drives and some of the terminals use the PDP's DMA facility and so it must be able to access the RAM at any time and without extreme delays.

Two possible choices for the TAC's DMA interface to the PDP are available: The first is to use a high priority DMA channel to transfer the frames to and from the system RAM; the second is to use some form of local

RAM situated on the interface card which can be accessed by both the TAC and the PDP 11/23 cpu.

The use of the local (or dual-port) RAM was prompted by the TAC's critical timing requirement and because there is no way of prioritizing the PDP's DMA transfer. Thus if another device was in the process of performing a DMA transfer, the TAC would have to wait for completion of the DMA cycle before it could use the bus. The dual-port RAM, on the other hand, allows the WD2840 to continue as a separate entity until a significant event occurs, such as token received, data frame received and frame sent. It then interrupts the PDP's cpu for service. (The WD2840 conviniently provides an interrupt line that can be used to generate this interrupt.)

A problem with dual-port RAM is that there is limited space on the I/O page - 8k bytes - which cannot all be dedicated to the LAN controller. Some means of mapping the RAM on the TAC board into a small section of the PDP's I/O page had thus to be found.

A second problem, related to the I/O page, was the accessing of the TAC's 16 status/control registers. As all the registers in the I/O page have to be on a word boundary, the 16 registers would require 16 words of I/O page which is excessive.

To reduce the I/O page address space requirement, the NIU is accessed via four registers. The first two are used to access the RAM while the second pair enable access to the TAC's status/control registers.

4.8.2 Description of the Implementation

A block diagram of the circuit is shown below. A detailed circuit diagram and the PAL logic functions may be found in appendix G.

The relevant Q-Bus timing diagrams are also given in appendix G of this report.

(a) The Control Interface

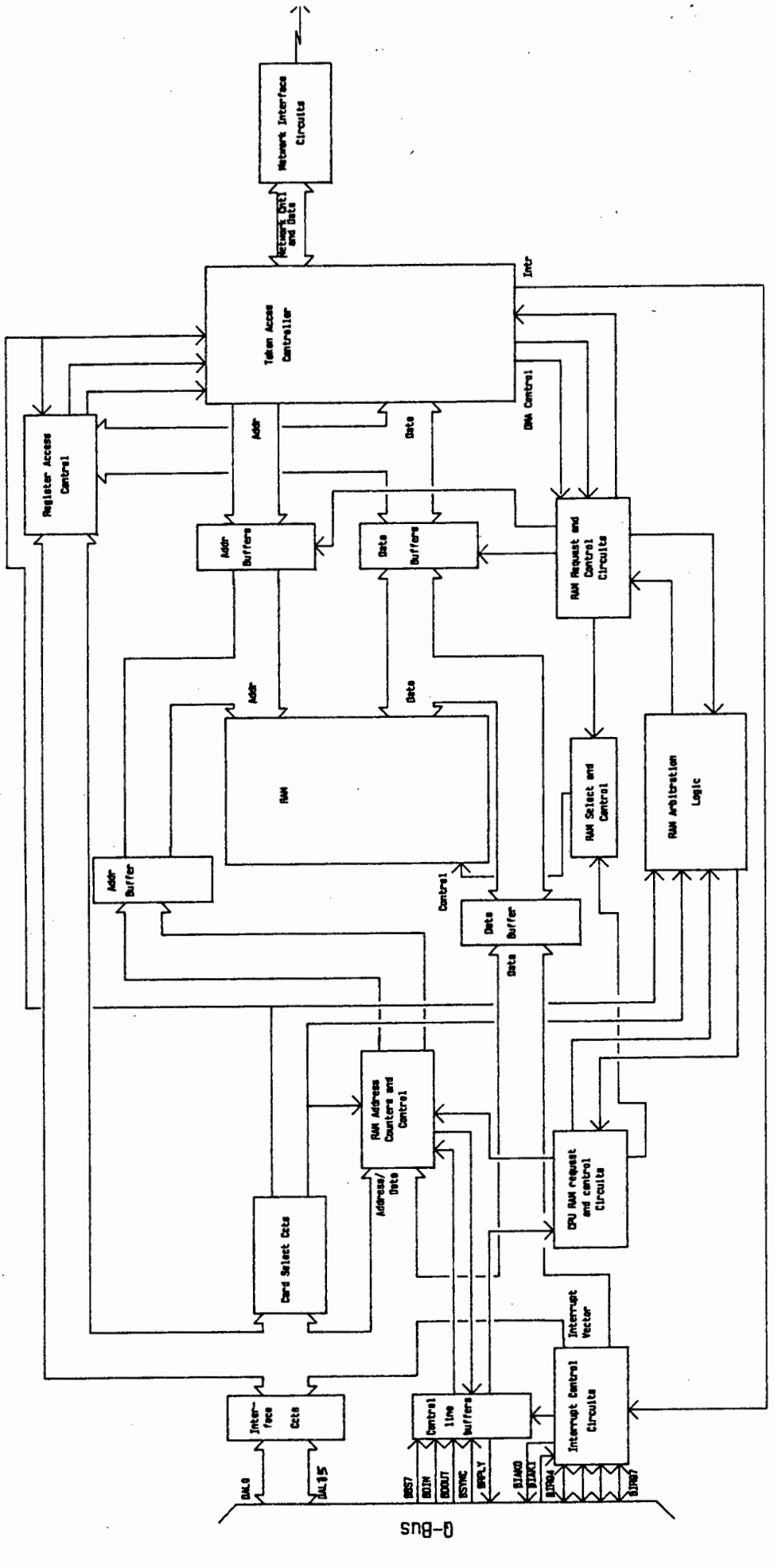
To provide versatility, the exact position in the PDP's I/O page of the NIU card, and therefore the 4 registers used to access the RAM and the TAC's registers, is mini-jumper selectable on the interface card.

Access to the TAC's registers is achieved by writing the address of the register to be accessed into the first of the four registers. This address is latched into a flip-flop until a new value is written to it. The addressed WD2840 register may then be read from or written to. In doing so the correct control lines, Chip Select (/CS) and Read Enable (/RE) or Write Enable (/WE), are activated under PAL control.

A shift register is used to enable the correct timing and to ensure that the WD2840 is not accessed more often than permitted, i.e. with less than 300 nanosec between chip selects.

(b) RAM Arbitration Logic

Some form of RAM arbitration had to be provided to enable the use of the RAM by both the PDP cpu and the TAC. This was done on a byte-by-byte, first-come-first-served basis with the TAC having the precedence should the two machines attempt to access the RAM at the same time.


Network Interface Circuits
Network Cntl and Data
Token Access Controller
Intr
Register Access Control
Addr
Data
DMA Control
Addr Buffers
Data Buffers
RAM Request and Control Circuits
RAM
RAM Select and Control
RAM Arbitration Logic
Addr Buffer
Control
Data Buffer
RAM Address Counters and Control
Address/ Data
CPU RAM request and control circuits
Card Select Data
Interrupt Vector
Interface Data
Control line buffers
Interrupt Control Circuits
DAL0
DAL15
BBSY
BDIN
BDOUT
BSYNC
BRPLY
BIAKO
BIAKI
BIRQ4
BIRQ7
Q-Bus

Because the RAM arbitration is done on a byte-by-byte basis, the TAC access is never delayed for an unacceptable period while the PDP's cpu accesses the RAM.

(c) Accessing the RAM

From the WD2840 side: once the arbitration logic circuitry has granted permission, the WD2840 RAM address buffers are enabled simultaneously disabling the PDP's buffers. The arbitration logic then sets a multiplexer to derive the RAM control lines (Output Enable, /OE, and Memory Write, /MW) from the WD2840 and not from the PDP's bus. The address is set up by the WD2840 via latches to ensure that it remains valid for the time required by the RAM. The chip selection of one of the four 2 kbyte RAMs is via a two-to-four line decoder, where address lines A11 and A12 are provided as inputs. During a write operation the data lines are latched to ensure that the data is maintained long enough for the RAM. The overall timing is controlled by a PAL.

From the PDP's cpu side: One method of reducing the I/O page requirement is to write the address to one register and then to read from or write to another register which simultaneously enables the address in the first, as in the case of the status/control register access above. But when the cpu accesses the RAM, it will generally access data in sequential addresses. To improve the efficiency, the "address register" is replaced by a 16-bit loadable counter. The cpu will thus first set up the starting address of the block of data by writing it to the address counter. The data may then be accessed via the second register without having to adjust the address register each time, as long as the access is sequential. A hardware requirement is that the address is one less than the first byte to be accessed as the counter is incremented at the start of each cycle.

Some versions of the PDP's cpu, specifically the version in the computer used to prototype the NIU card, use a read-modify-write cycle for their write cycle. To ensure that the address counter was not incremented twice, the top address line, A15, was set or cleared to indicate whether the following cycle was a read or a write respectively.

Once the arbitration logic had granted the cpu permission to access the RAM, the RAM was accessed as described above for the WD2840. The control lines to the RAM and the bus response lines are controlled by a PAL.

(d) Interrupt Circuitry

The WD2840 supplies an interrupt pulse whenever a significant event occurs. This pulse is latched onto the bus until the cpu responds. The priority of the interrupt is mini-jumper selectable as is the interrupt's vector address. The required NIU responses are under PAL control.

(e) Bus Interface Circuitry

The bus interface section of the circuitry is somewhat unusual because the bus specification requires high impedance input buffers and open collector drivers. This interface is therefore made up of a combination of two octal receivers and four quad open-collector drivers, all of which are controlled by a PAL.

In order to enforce the inter-access time, the Q-Bus's Reply control line is delayed via a shift register. This delay allows the WD2840 to update the transmission specific event counters in RAM as well as its internal registers.

(f) Reset Circuitry

The WD2840 may be reset by two sources. The first is the PDP's Q-Bus as a result of the cpu asserting the "Bus initialize" line. The second source is a manual reset on the card itself, which may be used during software debugging. Both initiate a monostable which provides the required length reset pulse to the WD2840 and the rest of the circuit.

(g) Clock and Network Interface Circuitry

The clock circuitry and the network interface circuit are the same as those used in the micro-computer NIU, as described in the previous section.

Chapter 5

The Network Interface Software

5.1 Introduction

This chapter continues the discussion on the implementation of the network related layers of the ISO RM. The subject is the software which provides the link between the hardware and the Transport/Session Layer interface, i.e. the information has been brought as far as the internal backplane of the computer by the hardware, it is now the task of the software to make it acceptable to the CP/NET software.

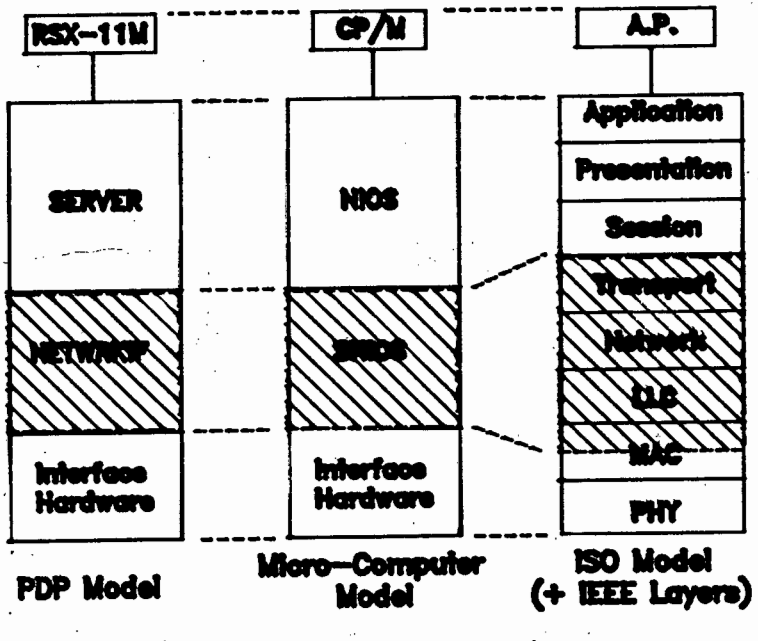


Figure 5.1 - Chapter Topic

The micro-computer software was not written specifically for CP/NET's NDOS because the interface may be used by students without the CP/NET programs. Thus a simpler interface was provided which would require only a

small adjustment to be compatible with NDOS. These transformation were done by the Token-bus SNIOS, also described below.

The software described for the PDP is simply test software which would form the basis of the device driver. At the end of the chapter the second part of the guidelines for the PDP software designer are given. These guidelines relate to the creation of the device driver to interface the PDP's executive to the LAN hardware described in the previous chapter.

5.2 Software Requirements

5.2.1 The LLC Layer Requirements

There are two types of service specified for the LLC/Network interface. The first is a connectionless service and the second a connection-oriented service. These two forms of service are used to define two types of LLC layers - those providing only a connectionless service, known as Type I LLCs, and those which provide both services, defined as type II LLCs. (The IEEE 802 states that a LLC layer may not simply provide a connection-oriented service.)

In a connection-oriented service, a virtual link is created between the two end nodes before transmission may begin. Each frame is sent along the same route and is acknowledged at the LLC level. At the end of the transmission session, i.e. when the link is no longer required, the link must be disconnected. A connectionless service, on the other hand, is one in which each frame is sent as a separate entity and has to mark its own route across the network. Here flow control and error recovery are provided by the higher layers.

In the IEEE 802 standards, the MAC Layer only provides a connectionless transmission which may either be point-to-point or multi-point and acknowledged or unacknowledged. The LLC layer is entirely responsible for creating the connectionless or connection-oriented service for the higher layers using this service.

If a connection-oriented service had been supplied by this LAN, the LLC Layer would first have had to set up the link between the two end points, where no intermediate nodes exist. The data packets would then have been sent along the link between the two nodes with the extra overhead bytes. Before an acknowledgement could be sent, the destination node would have had to have data to send or would have had to create a special acknowledgement frame.

Sequencing is not a problem on a LAN as the two most remote nodes will generally be less than the frame transmission time apart, i.e. the destination node would have begun receiving the frame before the source node had completed transmission. There are also no alternative routes which would delay one frame more than another. In this LAN, the WD2840 will implement flow control. It would seem unwise to duplicate this service in the higher levels of software. The act of creating and terminating the logical links would also lead to large data overheads when short interactive traffic is predominant.

The provision of a connectionless service appears more practical as:

1. Sequencing is not a problem on a LAN, the nodes are so close.
2. Flow control is provided by the TAC.
3. Each transmission on the LLC Layer is independent thus removing the need for link establishment and disconnection, and consequently reducing overhead.
4. A MAC Layer frame acknowledgement, provided by the TAC, means that this service can ensure frame arrivals.

Connectionless transmission was therefore chosen for this LAN, classifying each LLC entity as a class I LLC. The services provided are to transmit a frame without waiting for an acknowledgement and, as an improvement, to transmit frames with an acknowledgement request.

5.2.2 Parameter Initialization

Before the TAC is used on the network a number of parameters must be initialized. IEEE defines them as:

1. The station's address

2. The address length, implicit in the address value (16 or 48 bits long).

3. The frame acknowledgement time.

4. Token hold time, which defines the maximum time that the TAC is allowed for transmission of frames before the token must be passed.

5. Ring maintenance rotation timer, defining the maximum time allowed for token to circumnavigate the logical ring.

6. The maximum time between solicits for new stations.

7. "In-ring desired", which is a flag set by the host indicating that TAC must attempt to enter the logical ring.

8. Maximum retry limit if the acknowledged connectionless service is used.

There are some deviations from the IEEE standard due the limitations of the WD2840: The address length is only 8 bits instead of 16 or 48 bits long. The maximum retry count is fixed to two by the TAC.

5.2.3 The TAC Buffers

Once the TAC is part of the logical ring on the network the software is responsible for the maintenance of the TAC's buffers. The software must

also be able to flag the higher layers when a data unit arrives, and similarly it should be able to initiate transmission when requested to by the higher layers.

The WD2840 uses a chained buffer structure. On start-up the receive buffer pointer in the Control Block, a section of memory set aside for use by the TAC's buffer control, is set to point at the start of the first receive buffer. All but the last receive buffer have their pointers, the first two bytes of the buffer, pointing at the address of the next buffer in the chain. The last buffer's pointer contains zero as a flag. When a data frame is received, the TAC places it into the next available buffer. The DONE bit in the Frame Status Byte (FSB), a byte in the buffer's control field, is then set and the next buffer pointer in the Control Block is updated. The host cpu is then responsible for reading the buffer, clearing its FSB and linking it to the end of the chain for re-use, by adjusting the previous buffer's pointer.

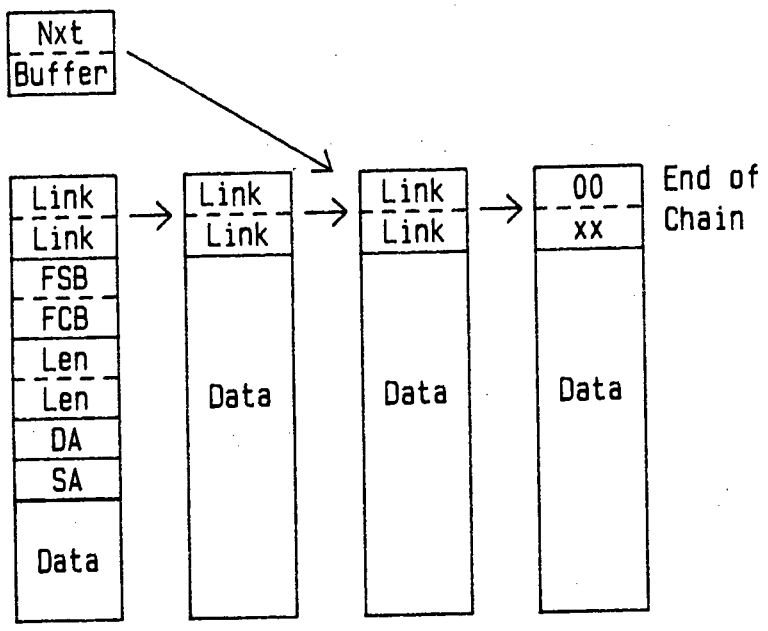


Figure 5.2 - The WD2840 Buffer Structure

The transmitter buffers are dealt with similarly. On start-up the Transmit buffer pointer in the control block is set to zero. When the first buffer is filled by the host cpu, this pointer is set to point at the

first buffer. As each subsequent buffer is filled, the pointer in the top two bytes of the previous buffer are changed from zero to the starting address of the new buffer by the software. Each frame contains a Frame Control Byte (FCB) which enables the control software to specify, amongst other things, whether an acknowledgement is required for that frame or whether this is the last frame. Once the WD2840 has transmitted, or attempted to transmit a frame, it writes one of the status values (given in Table 5.1) to the FSB; after the control software has read the status, the pointers are cleared and the buffer is made available for re-use:

FSB Value (in Hexadecimal)	Explanation of the Error
1	Insufficient receiver buffers
2	Receiver not enabled
3	Receiver over-run
4	Frame exceeded 16 receiver buffers
9	Transmission failure after retry
A	Transmitter under-run
B	Too little data supplied by host
C	Frame exceeded 16 transmit buffers

Table 5.1 - Returned Status Codes

A limitation of the WD2840 is that the Transmit and Receive clocks must not be more than 43% of the clock input to the TAC (which has a maximum of 2.05 MHz) if a single MAC frame is to consist of more than one buffer. By using 320 byte buffers, the largest frame required by the network software package, CP/NET, could be placed into a single buffer. To prevent failure in the event of a larger Network frame being passed to the LLC Layer for transmission, e.g. when CP/NET is not in use, the software segments the frames into 311 byte blocks. (The buffers are 320 bytes long, which includes 8 overhead bytes. Should the total number of data bytes

plus overhead bytes equal 320, the WD2840 will expect a second buffer even though it would contain no data.)

5.2a The Encapsulation of the Messages

Before launching into a description of the software it would be useful to show how the message is encapsulated on the different layers. Each layer was not implemented as a separate entity, e.g. the NDOS performs the Application, Presentation and Session layer functions, thus the interfaces are:

(a) The Application Process/Application Layer interface: At this interface a function is requested, e.g. login or read next sector. The user/application process will supply the function code and variables, in the case of "login" he will give the server's address and the password. The NDOS builds up a message which includes the source and destination addresses, the FNC and the SIZ fields.

(b) The Session/Transport Layer interface: The interface data unit at this interface is as follows:

FMT
DID
SID
FNC
SIZ
Data Byte 0
:
:
:
Data Byte n

The FNC and Data Bytes are the encapsulated data from the application process.

(c) The LLC/MAC Layer interface: The message is then converted into the form required by the hardware control program. At this interface the message looks like this:

Length High
Length Low
Destination Address (DA)
Data bytes

The DA is a copy of the DID provided above, and the data bytes contain the whole message passed across the Session/Transport Layer interface.

(d) The physical network interface: The WD2840 uses the standard HDLC format. This means that the message has been inserted into a frame to facilitate error recovery. The frame is as follows:

Flag (7EH)
Token Control (TC)
Destination Address (DA)
Source Address (SA)
Data bytes
Cyclic Redundancy Check byte
Flag

5.3 A Description of the SABus NIU Software

The SABus NIU software was written in 8085 assembler code and consisted of one program - the TAC User Routines (TACUR). This program was divided into five sub-sections, namely, network initialization, message transmission, message reception, status reporting and station isolation.

To provide easy access to the user, the routines may be run by calling one of five addresses at the beginning of the TACUR program. The entry point to the initialization routine would be at the start of TACUR, the entry point for message transmission at an offset of three from the start of the program, and the other routines at subsequent three byte intervals. (The program listing is given in appendix E of this report.)

5.3.1 The Initialization Function

The INITialization routine starts by setting up the parameters required by the TAC. The TAC is then instructed to enter the logical ring. If there is network activity, the NIU checks for a duplicate address (a function provided on the WD2840). If there is no address duplication, the TAC attempts to enter the logical ring at the next opportunity by responding to a token passed to it which will be during a solicit successor phase. The maximum wait is 1 second - the inter-solicit time.

If there is no activity on the network and the TAC has the right to claim the token, it attempts to establish the logical ring by continually polling the whole address space until another node responds. The TAC then informs the user, via the console, that it is part of the network or that it has the same address as another active node, and returns to the calling program.

5.3.2 The Transmission of Frames

The second function, the transmission of frames across the network, is performed by the SNDMSG routine. Two flags are used to indicate the required service. The first is the accumulator which requests transmission with or without waiting for acknowledgements. The second is the 8085's B-register which indicates whether the data buffer contains a command or a non-command data unit. If B is set to 0, the frame in the data buffer is an Unnumbered Information (UI) frame, in which case the routine adds the IEEE UI Control byte. If the value in the B register is 255 then the data buffer contains a control frame and the SNDMSG routine will not add the UI Control byte as the frame contains its own control byte.

The data is transferred to one of the two WD2840 transmit buffers. Once the buffer header has been set up, the buffer is linked to the previous TAC transmit buffer and transmission is enabled. This routine then waits until the WD2840 signals that it has completed the transmission. The value in the FSB is then read to establish whether or not the transmission was successful. If an error occurred, one of the status codes (given in Table 5.1) is returned in the accumulator, as well as the start address of the last block of data in which the erroneous transmission occurred, contained in the DE register pair. This is to facilitate recovery by the higher levels.

If no error occurred and if there is more data to send, the SNDMSG routine transmits the next 311 byte block until either an error occurs, or the input buffer is empty and then it returns.

5.3.3 The Reception of Data Frames

Because the micro-computers are single user machines, the reception of data from the WD2840 is controlled by polling rather than via interrupts. On entry to the receive message routine (RECMSG), the program first establishes if a data frame has been received. If not, it delays 10 msec, to enable the TAC to update its registers, and then returns with the

accumulator set to 0. If there is data, the RECMSG routine reads the frame from the TAC receive buffer into memory.

The IEEE control word is then compared to one of the supported functions, namely Exchange Identification (XID), TEST and Unnumbered Information (UI). If the frame is an XID, the RECMSG sends an XID frame which is stored in memory. If the frame is a TEST frame, the received data is looped back to the transmitter. In both the above cases, the Final bit (a bit in the IEEE control byte) is set to the same value as the Poll bit in the received control byte. If the illegal condition of a UI frame with the Poll bit set is received, the frame is read in from the buffer and discarded. In each of the above cases the accumulator is set to zero to indicate that no data was received on procedure return. Finally, if the received frame is a UI frame with the Poll bit cleared (i.e. a valid data frame), the data is read from the TAC input buffer to the output buffer pointed to by the DE register pair. The value of the accumulator is then set to 255 indicating that data was read and the procedure returns.

5.3.4 The Status Report Function

The fourth function provided by the TACUR program is a status report performed by the STAT routine. This routine reads the values of the 16 TAC status/control registers as well as the 11 event counters (which give a count of the occurrence of a number of non-fatal network related events). Each of the counters is set to 0 as it is read. It then returns these values in a buffer pointed to by the DE register pair. This status read enables the application program to calculate the best values for the WD2840 parameters and adjust them while the TAC is on-line.

5.3.5 The Isolate Function

Finally, a function which enables the removal of the TAC from the logical transmission ring is provided. This routine instructs the WD2840 to leave the network. A message is printed to the console and the accumulator

returns a flag to indicate whether the isolation is complete or whether confirmation from the TAC is still pending.

5.4 The Token-bus SNIOS

This program performs the transformation between the standard interface provided by the TACUR program and the interface as required by the CP/NET NDOS program. Its listing may be found in Appendix F of this report.

The TAC User Routines program, described above, provides the user with a LLC/Network layer interface. The SNIOS simply has to transform this interface into one that is acceptable to the NDOS. The overall structure of the SNIOS program is given in the CP/NET Users Guide [14].

The first function of the SNIOS is to initialize the network which is done by calling the TACUR initialize routine. To send a message, SNIOS calls the send message routine in TACUR after adjusting the packet format to conform to the TACUR format. Similarly the reception of a packet is achieved by calling the receive message routine in TACUR and adjusting the format of the packet received.

5.5 The Requirements of the PDP NIU Software

There are two levels of software that make up the device driver. The first level (the only level dealt with in this project) controls the transmission and reception of data frames and the NIU. The second level is responsible for accepting instructions from the RSX-11M Executive and acting on them to provide multi-user access to the network. This level of the software is left for the writer of the PDP 11/23 Network Server Software.

The software used to drive the PDP LAN card is written in LSI-11 assembler code. Because the PDP is used by other students and staff, a

FALCON single user computer, which uses a similar processor and also has the Q-Bus as its internal bus, was used for hardware and software development. Once the higher layers of software are written, the interface card may be plugged directly into the PDP 11/23.

The first requirement of the program is to initialize the hardware according to the IEEE standard as detailed above. The software must then control the TAC, provide correctly formatted buffers, instruct the TAC to send the frames and read frames from the RAM. In order to test the network, some form of user interface had also to be provided. Finally, some code had to be written to service both the Line Time Clock (a real time clock) and WD2840 interrupts.

5.6 A Description of the PDP NIU Software

The program listing may be found in appendix H of this report. It may be broken into 5 sub-sections: control and user interface routines, the device initialization routine, the frame transmission routine, the reception routine and the interrupt service routines.

5.6.1 User Interface and Control Routine (MAIN)

The MAIN routine, which may be thought of as the station management, has overall control of the program. It begins by calling the initialization routine to set up the parameters required by the TAC. The initialization routine instructs the TAC to go on-line and attempt to enter the logical loop. When it returns to the MAIN routine, the user is informed of the status and is prompted to select either data transmission or reception.

While the computer is waiting for a response from the user, it continually checks to see if a data frame interrupt has been received. If an interrupt has occurred, or if the user chooses to receive a frame from the network, the MAIN routine calls the message receive (RECMSG) routine.

Once the TAC has completed reception, RECMSG reads the TAC's buffer into a temporary input buffer which is used to display the message on the console.

If the user chooses to send a frame, he is prompted for the destination address and the message, SNDMSG, is then called. This routine sets up the buffers, links them to the previous ones and enables TAC transmission. Once transmission of the frame is complete, it returns a status report to MAIN procedure.

5.6.2 Initialization Routine (INITNET)

This routine has the task of including the TAC in the logical ring on the network. It first sets up the WD2840 registers and then checks for network activity by sensing the LAN for a fixed interval defined by the Line Time Clock. If no Network Dead interrupt occurred during this period, the network is presumed to be active. If this is not the case, the TAC is instructed to attempt to start up the network by initiating the scanning procedure. The TAC continues scanning until it receives a response from another TAC.

On an active network, a check is made for a duplicated station address. This is done by setting a timer to the solicit successor interval. If only one token interrupt occurs in this period, it is assumed to be as the result of a solicit. Should more than one interrupt occur, an active station must have the same address as the station attempting to enter the logical ring on the network. More than one interrupt would occur in the duplicated address situation as the active station is addressed each token rotation, which must be within the token rotation time, which, in turn, is less than the inter-solicit interval. In the duplicate address case the TAC is disabled and the station management informed.

Once it has been established that there is no address duplication, the TAC attempts to enter the logical ring on the network, i.e. the TAC will respond to the next token addressed to it. Once the logical loop is established, INITNET returns to the MAIN routine.

5.6.3 Transmission Control Routine (SNDMSG)

The SNDMSG routine controls the transmission of frames. It first prompts the user for both the destination address and the message, which is limited by the software to 650 bytes. Once the message has been read, the FILTXBUF routine, which transfers the data from an internal buffer to one of the TAC's buffers in dual-port RAM, is called to create a frame and send it to the TAC transmit buffers (located in the dual-port RAM) in the required format. The FILTXBUF routine then instructs the TAC to send the frame and waits for its completion. This routine then returns the transmission status in the R1 register. The returned status values are the same as those used in the SABus software, as listed in Table 5.1.

The FILTXBUF routine is written so that it can be used in the device driver and so makes no assumptions about the size of the transmission buffer. It segments the buffer received into frames of 311 bytes long and adds the IEEE control byte specified in the CONTTYPE variable to each frame.

The SNDMSG routine waits for FILTXBUF to return and then returns to the MAIN routine.

5.6.4 Receive Message Routine (RECMSG)

This routine checks for a full receive buffer. If none is found it returns. Should a buffer be available, it clears both the FSB and the appropriate bit in the interrupt register and then calls the Read TAC Receive Buffer (READWDBUF) routine in order to transfer a frame from one of the TAC's buffers to an internal buffer for screen display.

The READWDBUF routine finds out which buffer should have been written to and checks the FSB for confirmation. If the DONE bit in the FSB is not set, either this buffer contains an aborted frame, which means that the next buffer may be full, or no data was received. If neither this buffer or the next has its DONE bit set, the READWDBUF routine returns with the

NODATA flag set to true. If the DONE bit is set, the READWDBUF routine reads the data to the input buffer (INBUF) and links the TAC buffers in dual-port RAM to the chain for re-use by the WD2840.

In order to ensure that all the buffers are read, the READWDBUF routine reads the FSB of the following two frames to see if the data contained in them is ready for access. If this is the case then the READWDBUF sets the NOMOREDATA flag to FALSE.

If the NODATA flag is FALSE the RECMSG routine writes the frame source address, from the first byte of INBUF, to the screen after converting it to two ASCII characters. The rest of the message in INBUF is then sent directly to the console. NOMOREDATA is checked to ascertain if READWDBUF must be called to read a second buffer. Once all the available frames have been read, RECMSG returns to the MAIN routine.

5.6.5 Interrupt Service Routines (LTCSR and WDSR)

These two service routines deal with interrupts from the two interrupt sources, the Line Time Clock and the NIU card.

(a) LTCSR

The Line Time Clock Service Routine (LTCSR) provides a means of calculating long delays, e.g. during duplicate address checking, and ensures, every second, that new successors are solicited.

To create a means of calculating long delays, the 20 msec hardware timer interrupt is enabled. The interrupt service routine simply increments a counter each time an interrupt occurs. The counter is then read by other sections of the software when long delays are required. After one second, i.e. 50 interrupts, the interrupt service routine initiates a scan procedure by setting the TAC's successor address to its own plus one.

(b) WDSR

Because the TAC is receiving frames from the network at 1 Mbps, multiple interrupts to the same interrupt routine may occur. On the PDP this would not cause failures as the interrupt service routine may increase its priority, thereby excluding further interrupts to the same location until service is complete. The FALCON computer does not permit interrupt priorities and so multiple interrupts become problematic. To overcome this, the interrupt handler was made as short as possible.

The Western Digital Service Routine (WDSR), which services any interrupts from the NIU card, is called when the WD2840 initiates an interrupt. The WDSR reads the value of the TAC's interrupt register (IR0) and sets the appropriate bit, or bits, in the IRDATA register in the PDP's RAM. The IRDATA register is read by the other subroutines to establish if a particular interrupt has occurred.

The IR0 register must be read twice in order to clear the interrupt because of faults in the prototype version of the WD2840.

5.7 Guidelines for the PDP Server Program (Part 2)

The PDP device driver forms the link between the interface hardware and the PDP's Operating System Executive (that section of the operating system which directs program execution). The purpose of the device driver is to control the interface hardware thereby allowing the operating system to perform a number of standard functions, e.g. reading and writing of blocks of data.

The device driver may be broken into 5 sub-sections, each requiring a separate entry point for the executive. These sub-sections are:

1. I/O Initiator
2. Device interrupt
3. Device timeout

4. Cancel I/O
5. Power failure

The I/O initiator is called to "wake-up" the device driver when a task queues one of the functions. For instance, to send a frame, a task would queue a write block request to the executive. The executive would then call the device driver at the I/O initiator entry point giving details of the function to be performed.

Once the device has completed the function it would interrupt the cpu which would call the Device Interrupt entry point. This section of the driver would thus have code to deal with device interrupts.

The Device Timeout entry point is called to cancel an I/O function which has been active too long.

The Cancel I/O entry point provides a point for the executive to call in order to cancel a previously initiated I/O function call. This is generally as a result of a task command rather than an executive timeout.

When power is restored and a device driver was running before failure, the Power Failure entry point may be called to initialize the device hardware. The executive may also be instructed to call this entry point when the driver is loaded.

Sections of the PDP's NIU software, described above, may be incorporated into the device driver. Firstly, the I/O initiator would have to be able to perform a number of functions, e.g. transmission and reception of blocks of data (or frames), device initialization and reading and adjusting certain parameter values. (The allowed functions must be specified within the driver software.) Three of the NIU software routines may be called by the initiator:

1. INITNET - to start-up the network.
2. SNDMSG - to initiate the transmission of frames.

3. RECMSG - to read a frame received from the network.

Secondly, the power failure routine could call INITNET to restart the interface hardware.

Chapter 6

Network Verification and Conclusions

This chapter examines the results gathered from tests performed on the network from which a number of conclusions are drawn.

6.1 Network Verification

The LAN was constructed as described in the previous chapters. Circuits were laid out on printed circuit board, constructed and tested. A test rig was then set up, as shown in Figure 6.1, to verify network functioning.

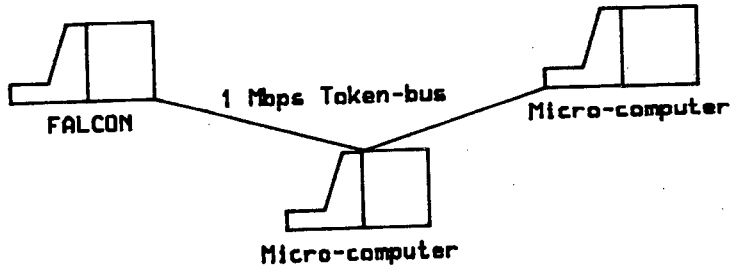


Figure 6.1 - The Token-bus LAN Test Rig

6.1.1 Functional Performance

Because the PDP Server program has not yet been written, three tests were necessary to confirm the correct functioning of the partially complete LAN.

Firstly, the implementation of CP/M and the CP/NET programs on the U.C.T. micro-computers was verified. This was done by connecting two of the computers via separate RS-232 links to an MP/M-II server. Using Digital Research software to control the network oriented layers, and thus provide serial communication, the U.C.T. micro-computers were able to successfully communicate with the MP/M Server and use the utilities provided. This verified CP/M because CP/NET loaded and ran correctly. All the CP/NET functions were used. They worked correctly, thus confirming correct implementation of the CP/M and NDOS part of CP/NET on the U.C.T. micro-computers.

Secondly, the implementation of IEEE layers was tested using a specially constructed network, as illustrated in Figure 6.1. Using the software described in chapter 5, messages of varied lengths and contents were successfully transmitted between the micro-computers and the Falcon. The inter-solicit timers were tested by instructing one of the computers off the network and then immediately back on again. The computer was included in the logical loop within the inter-solicit time of 1 second. It was discovered that the duplicate address checking facility provided on the WD2840 prototype versions did not function correctly. A duplicate station address thus led to the exclusion of one of the stations with a duplicated address. This portion of the program was not adjusted as it would function correctly with future versions of the TAC.

Finally, the network as a whole and its interaction with CP/NET was tested by allowing the micro-computers to issue CP/NET requests to the Falcon, which was acting as the Server. Manual Server responses were provided from the Falcon's keyboard. In all cases the network was found to function correctly.

6.1.2 User Level Software

Briefly the user level requirements for the training network were to provide:

1. Program capture
2. Assembly and debugging of assembler code
3. Compilation of high-level languages programs
4. File storage and recovery
5. Electronic Mailing and Word processing facilities

The above requirements were implemented by the combination of CP/NET and CP/M on the micro-computers and will be provided by the software on the PDP.

To facilitate program capture a CP/M editor, e.g. MicroSoft's WordMaster, could be used. This program was run over the RS-232 test network and performed correctly. Other CP/M programs could provide the assembler and debugging facilities, e.g. ASM for the assembler facility and SID or DDT as a debugger. Any of these programs could run under CP/M on the micro-computers. There were also a number of high-level languages supported by CP/M, e.g. PASCAL MT(, FOURTH, PL/M and many others, which could be used on the network.

The assembler debugging program, SID, was tested during the token-network development phase. This program was run from the EPROM disk. The Electronic Mailing facility is part of CP/NET and was also tested with the help of the RS-232 test network. File storage and recovery were tested in the same way. A word processor, WordStar, was also run over the RS-232 network.

6.1.3 Physical Performance

The physical requirements for the network were:

1. It should span 800 m.
2. It had to support up to 30 terminals.
3. A speed requirement of 1 Mbps.

Figure 6.2 shows the data signalling rate versus cable length for a CCITT V.11 transmission (an equivalent of the EIA RS-422 standard). It is clear that even for a terminated cable (curve 1), 800m is too long. To achieve this distance, the 800m length was broken into 75 m sections interconnected by bi-directional RS-422 repeaters.

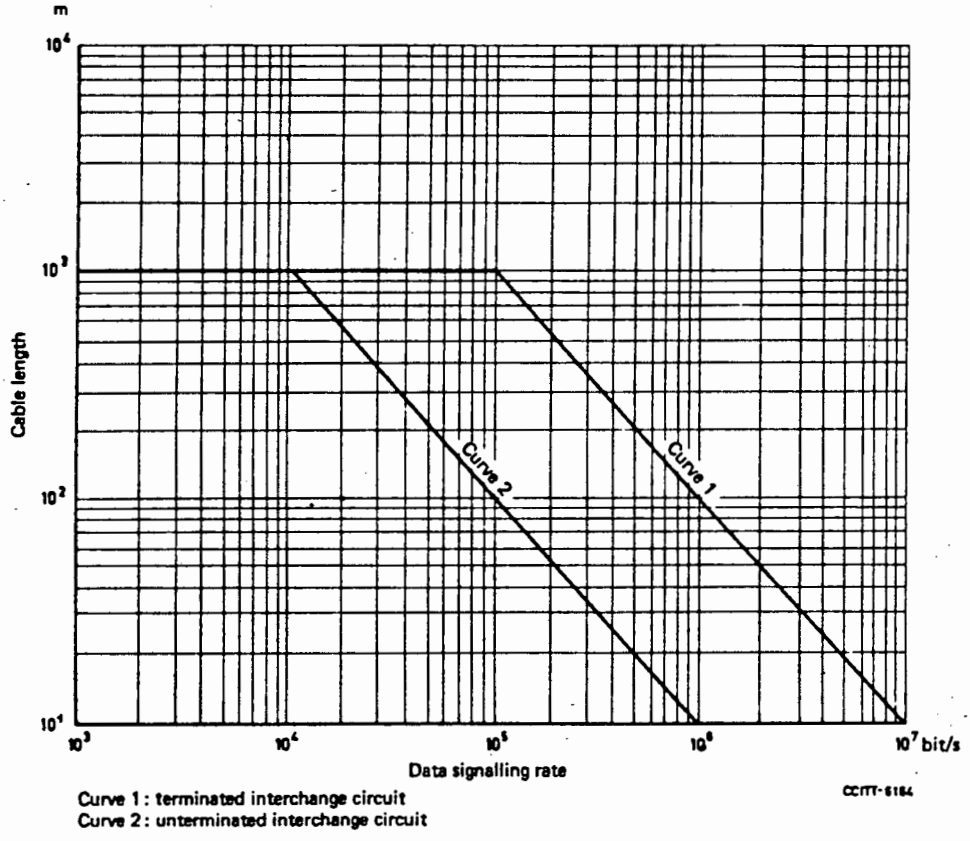


Figure 6.2 - Data Signalling Rate versus Cable Length for Balanced Transmission

To ensure correct transmission over 75 m, a distance test was performed by connecting a 25 m section of 3 pair cable between the two micro-computers in the test rig. The cable was connected so that the signal travelled 75 m by joining two pairs at each end. The signal travelled the required 75 m in a noisy environment, inducing noise on itself, without an increase in the frame retransmissions due to noise.

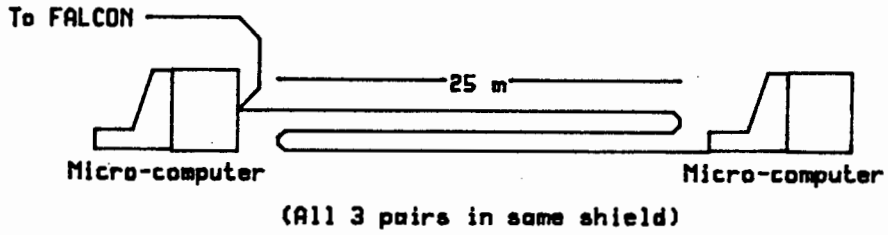


Figure 6.3 - Cable Connections for the Distance Test

The second requirement could not be tested as there were not 30 terminals or line drivers available. The specification for the RS-422 driver used states that 32 drivers can be used on a common link. The software addressing makes provision for 32 nodes on the bus and so, although this feature was not tested, if the chip performs according to specification, the requirement will have been fulfilled if the chip is found to perform according to specification.

6.1.4 Data Traffic Performance

Because the PDP server software is not complete, it is difficult to verify the total network performance. The LAN uses a 1 Mbps transmission rate and so the file transfer rate will be far higher than that of the original network. As far as queuing delays are concerned, it can only be said, at this stage that the file transfer time for a 44k byte frame is about 2 seconds on the test rig, with two micro-computers transmitting simultaneously. The PDP software author will have to provide an access fast enough to keep the delay to a minimum.

6.1.5 Administration Parameters

The TACUR program provides a status report which supplies the following parameters:

1. The number of scan frames received.

2. The number of transmission failures on first attempt which were successful on the second attempt.
3. The number of erroneous access control frames received.
4. The total number of Negative Acknowledgements sent.
5. A count of the invalid frames received due to abnormal network conditions.
6. The number of duplicate tokens received.
7. A duplicate address count.
8. The number of aborted transmissions.
9. The number of times the network has been inactive for the timeout period.

These values, in conjunction with a real time clock and a software counter to count the number of frames sent and received, can be used to supply a variety of data for monitoring and control purposes. The software will form part of the PDP server software.

6.1.6 Future Requirements

The network should be able to expand to accommdate a second mini-computer, voice and video data and the connection of other terminal equipment.

The second mini-computer could be included by using a version of the network interface hardware and the driver software written for the PDP.

Should the baseband transmission be replaced by the FSK broadband transmission, and the voice and video interface circuits are constructed, this network could certainly support multiple data channels over a coaxial cable bus.

Any data terminating equipment may be connected to the network. To achieve this a version of the network interface unit would have to be constructed, and the DTE would have to be "intelligent" enough to drive it.

6.2 Compliance with the IEEE 802 Standard

The only section of the project that complies with the IEEE 802 standards is the LLC layer, which implements a Class I LLC layer.

The MAC layer does not comply to the IEEE 802.4 standard because of the use of the WD2840 TAC. The TAC was used because of its low cost, availability and ease of use.

On the PHY layer, the network was originally designed to comply with a IEEE 802.4 specification for Phase-continuous FSK. This design was based on the use of a pair of FSK modem chips which were not produced in time for use in this project. The PHY layer was thus made to consist of a bus of twisted pair wire which is driven by RS-422 drivers, which does not comply with the IEEE 802.4 standard.

6.3 Conclusions

The introductory chapter high-lighted three reasons for the development of a LAN in the Department of Electrical and Electronic Engineering at U.C.T., these were:

1. The old network was slow.
2. It was restricted in its switching ability.
3. High level software was too expensive, and, if used, would overload the PDP even more.

In principle, it is best to adhere precisely to an international standard, but the design for each LAN is determined by the requirements for that network. To adhere precisely to the IEEE 802 standards in this case would have been too costly, time consuming and complex. By following the principles of the 802 standards and by using the WD2840 network controller, the beginnings of an inexpensive yet satisfactory LAN were created.

The 1 Mbps Token-passing bus successfully overcomes the three drawbacks of the original network and makes provision for the supply of a micro-computer training facility for a number of years.

References

1. SHERLOCK, B.G., "A Multi-user Interface System for a Process Control Computer", Engineering Masters Dissertation, U.C.T., 1983

2. EVA, J.A., "A Multi-user Microprocessor Assembler Language Development Environment", Engineering Masters Dissertation, U.C.T., 1986

3. JACK J.G., Engineering Masters Dissertation, U.C.T., to be submitted 1987

4. ABRAMS, M.D., "Observations on Operating a Local Area Network", COMPUTER, pp 51-65, May 1985

5. TANENBAUM, A.S., "Computer Networks", Prentice/Hall International, pp 286-288, 1981

6. FRITZ, J.S, KALDENBACH, C.F., POGAR, L.M., "Local Area Network Selection Guide", Prentice/Hall Englewood Cliffs, pp 4-7, 1985

7. SABUS COMMITTEE, "SABUS - The South African Standard Bus for Microprocessors", Pulse, pp 49-50, Nov 1981

8. DIGITAL EQUIPMENT CORPORATION, "Micro Computer Processor Handbook" Data Sheet, 1980

9. DIGITAL RESEARCH, "CP/NET - Network Operating System Reference Manual"

10. SHORT, K.L., "Microprocessors and Programmed Logic", Prentice-Hall Inc., Englewood Cliffs, pp 19-21, 1981

11. PRATT, S.J., "Distributed Computing: Considerations for its use within Educational Environments", Computer Education, Vol. 9, No. 4, pp 197-204, 1985

12. WILLIAMS, M.H., SARTORI-ANGUS, A.G., STROMBERG, G.P., "Alternative approach to a microprocessor teaching laboratory", Microprocessors and Microsystems, Vol. 4, No. 6, pp 219-222, Jul/Aug 1980

13. HUTCHISON, D., CORCORAN, P., "A microprocessor-based local network access unit", Microprocessors and Microsystems, Vol. 6, No. 1, pp 3-7, Jan/Feb 1982

14. DIGITAL RESEARCH INCORPORATED, "CP/NET - Network Operating System Reference Manual"

Additional Reading List

Below is a list of references that provided a background for the development of this LAN:

On the subject of other similar LANs:

15. AYLOR, J.H., WHITE, E.J., "Creating a Microcomputer Facility", IEEE Transactions on Education, Vol. E-24, No. 1, pp 47-50, Feb 1981

16. COLIN, A.J.T, HUTCHISON, D., SHEPHERD, D., "Microprocessor Teaching Laboratory", Microprocessors and Microsystems, Vol. 3, No. 4, pp 169-172, May 1979

17. GLOVER, J.R., BARGAINER, J.D., "Integrated Hardware and Software in a Computer Engineering Laboratory", IEEE Transactions on Education, Vol. E-24, No. 1, pp 22-27, Feb 1981

18. HANSON, D.F., "A Microprocessor Laboratory fo Electrical Engineering Seniors", IEEE Transactions on Education, Vol. E-24, No. 1, pp 8-14, Feb 1981

19. MCCREDIE, J.W. (Ed.), "Campus Computing Strategies", Digital Press, Massachusetts, 1983

20. THOMAS, D.E., "A Laboratory Environment for the Introduction of Microprocessor Systems into the Electrical Engineering Curriculum: Methodology and Experiences", IEEE Transactions on Education, Vol E-22, No. 2, pp 105-107, May 1979

On the subject of LAN performance:

21. ACAMPORA, A.S., HLUCHYJ, M.G., "A New Local Area Network Architecture Using a Centralized Bus", IEEE Communications Magazine, Vol. 22, No. 8, pp 12-21, Aug 1984

22. BUX, W., "Local-Area Subnetworks: A Performance Comparison", IEEE Transactions on Communications, Vol. COM-29, No. 10, pp 1465-1473, Oct 1981

23. BUX, W., CLOSS, F.H., KUEMMERLE, K., KELLER H.J., MEULLER, H.R., "Architecture and Design of a Reliable Token-Ring Network", IEEE Journal of Selected Areas in Communications, Vol. SAC-1, No. 5, pp 756-765, Nov 1983

24. KONHEIM, A.G., MEISTER, B., "Waiting Lines and Times in a System with Polling", Journal of the A.C.M., Vol. 21, pp 470-490, 1974

25. LAM, S.S., "A Carrier Sense Multiple Access Protocol for Local Networks", Computer Networks, Vol. 4, pp 21-32, 1980

26. MURRAY, D.N., ENSLOW, P.H., "An Experimental Study of the Performance of a Local Area Network", IEEE Communications Magazine, Vol. 22, No. 11, pp 48-53, Nov 1984

27. SALTZER, J.H., POGRAN, K.T., CLARK, D.D., "Why a Ring?", Computer Networks, Vol. 7, pp 223-231, 1983

28. STALLINGS, W., "Local Network Performance", IEEE Communications Magazine, Vol. 22, No. 2, pp 27-36, Feb 1984

29. STUCK, B.W., "An Introduction to Traffic Handling Characteristics of Bus Local Area Network Distributed Access Methods", IEEE Communications Magazine, Vol. 22, No. 8, pp 46-56, Aug 1984

APPENDIX A

The BIOS Program for the UCT Micro-computers

```
        title 'UCT BIOS adapted for Mass Storage Card Access'
        page 35
;*****************************************************************************
;*                                                                           *
;*                        BIOS FOR UCT SABUS KITS                            *
;*                                                                           *
;*****************************************************************************
;
;               PROGRAM  : KBIOS.ASM
;               LANGUAGE : 8085 Assembler Code
;               AUTHOR   : Adapted from Microcom Bios by Q.P. Mc Grath
;               DATE     : 9 May 1985
;
; KBIOS is the modified version of the CP/M BIOS to enable the use of CP/M in
; the U.C.T. built SABus Kits.
;
; The KBIOS may be devided into the following sections:
;
;       1) The jump table
;       2) The disk definitions
;       3) The boot routines
;       4) The disk package
;       5) The communications package
;       6) The communications variables
;       7) The disk variables
;
; 1. The jump table is the entry point for the system calls to the program.
;
; 2. The disk definitions are to enable the BIOS to access the disk, as it
;    needs to know the format of the data.
;
; 3. The boot routines are there to enable the system start up.
;
; 4. The disk package is the section that deals directly with the reading
;    of the data from the Mass Storage Card.
```

```
;
; 5. The communications package, this section provides a tool for communicating
;    with any type of terminal and printer. The package enables the use of the
;    console port, USART 1 and the parallel port, in a variety of formats.
;
; 6. Communication varibles,
;
; 7. Disk varibles, for the storage of varibles.
;
;-----------------------------------------------------------------------------
        page
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*                          EQUATES                              *
                ;*                                                               *
                ;*****************************************************************
                ;
                ;       BIAS is address offset from 3400H for a memory system other
                ;       than 16k.
                ;
0036 =          MSIZE   EQU     54              ; 54k system
8800 =          BIAS    EQU     (MSIZE-20)*1024 ; The extra bias required
B500 =          CCP     EQU     0DD00H - (64-MSIZE)*1024 ; CCP base address
BD06 =          BDOS    EQU     CCP+806H        ; BDOS base address
CB00 =          CBIOS   EQU     CCP+1600H       ; CBIOS base address
0004 =          CDISK   EQU     0004H           ; Current disk storage space
0003 =          IOBYTE  EQU     0003H           ; The IO Byte address
                ;
                        page
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*                    SECTION 1 - BIOS JUMP TABLE                *
                ;*                    ===========================                *
                ;*                                                               *
                ;*****************************************************************

                ;
                ;       Jump vectors for CBIOS subroutines
                ;
0000 C30301             JMP     BOOT            ; Cold boot entry
0003 C37F01     WBENTRY JMP     WBOOT           ; Warm boot entry
0006 C3E601             JMP     CONST           ; Console status
0009 C3F001             JMP     CONIN           ; Console input
000C C3FA01             JMP     CONOUT          ; Console output
000F C30402             JMP     LIST            ; List output
0012 C31002             JMP     PUNCH           ; Punch output
0015 C31C02             JMP     READER          ; Reader input
0018 C38401             JMP     HOME            ; Home drive
001B C39001             JMP     SELDSK          ; Set disk
001E C3A101             JMP     SETTRK          ; Set track
0021 C3AD01             JMP     SETSEC          ; Set sector
0024 C3BB01             JMP     SETDMA          ; Set DMA
0027 C3C301             JMP     READ            ; Read the described sector
002A C3E001             JMP     WRITE           ; Write the described sector
002D C32602             JMP     LISTST          ; List device status
0030 C3E301             JMP     SECTRAN         ; Sector translate

                        page
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*                  SECTION 2 - DISK DEFINITIONS                 *
                ;*                  ============================                 *
                ;*                                                               *
                ;*****************************************************************
                ;
                DSKHED:                             ; Disk header definition
0033 0000               DW      00H                 ; Translation vector address (0=none)
0035 31000200           DW      0031H,0002H         ; Scratch area
0039 8000               DW      0080H               ;
003B 4300               DW      DIRSCR              ; Directory scratch area
003D F400               DW      DSKDEF              ; Disk definition Block pointer
003F C300               DW      CSCR                ; Check sum scratch area
0041 D300               DW      ALV                 ; Allocation information
                                                    ;
0043            DIRSCR: DS      128                 ; Directory scratch area
00C3            CSCR:   DS      16                  ; Check sum scratch area
00D3            ALV:    DS      33                  ; Allocation information
                DSKDEF:
00F4 4000               DW      64                  ; 64 Sectors per Track
                                                    ;   Here Sectors*128=Address on the eprom
                                                    ;   while the Track is the Eprom number
00F6 04                 DB      4                   ; BSH - Block shift factor
00F7 0F                 DB      15                  ; BLM, Together with BLS gives a block size of
                                                    ;      2k long.
00F8 00                 DB      0                   ; EXM - The extent mask, determined by the
                                                    ;    total number of blocks
00F9 0001               DW      256                 ; DSM - The maximum number of blocks on this
                                                    ;     'drive'.
00FB 2700               DW      39                  ; DRM - The maximum number of directory entries
00FD 80                 DB      10000000B           ; AL0 - With AL1 indicates the number of
00FE 00                 DB      00000000B           ; AL1 -   blocks reserved for directory entries
00FF 0000               DW      0                   ; CHK - Giving the number of checked directory
```

```
                                        ;       entries.
0101 0100               DW      1       ; OFF - Indicating the number of reserved
                                        ;       tracks at the start of the disk
                ;*****************************************************************
                ;*                                                               *
                ;*                    SECTION 3 - BOOT ROUTINES                  *
                ;*                    =========================                  *
                ;*                                                               *
                ;*****************************************************************
                ;-----------------------------------------------------------------
                ;                                                                £
                ;       FUNCTION: BOOT;                                          £
                ;          Inputs:      None                                     £
                ;          Outputs:     None                                     £
                ;          Destroys:    A, B, C, D, E, H, L, Flags               £
                ;          Calls:       WBOOT,CONOUT                             £
                ;          Description: This routine gets called by the cold start loader £
                ;                       which had loaded CPM. IOBYTE is initialised and £
                ;                       the signon message is displayed. The blocking £
                ;                       parameters are initialised and the entry points £
                ;                       of the of the operating system set. The cold £
                ;                       start autoload is placed into the CCP and the CCP £
                ;                       is called.                               £
                ;                                                                £
                ;-----------------------------------------------------------------
                ;
0080 =          OEPNUM  EQU     80H             ; The old EPROM number
0094 =          INTIOBY EQU     10010100B       ; LST=LPT, PUN=PTP, RDR=PTR, CON=TTY
001B =          ESC     EQU     1BH             ; The ASCII ESC value
B507 =          COMLINE EQU     CCP+7H          ; Command line for auto entry
                ;
                BOOT:
0103 3A8000             LDA     OEPNUM          ; Read the old number
0106 3D                 DCR     A               ;   decrement by one
0107 328B03             STA     EPOSET          ;   and write it to the new position
```

```
010A CD8702            CALL    BOOTCOM         ; Set up the Communications
010D 0621              MVI     B,MESLEN        ; Send the Sign on message
010F 215E01            LXI     H,MESSAGE       ;
               MESLOOP:                        ;
0112 E5                PUSH    H               ;
0113 C5                PUSH    B               ;
0114 4E                MOV     C,M             ; Read the Byte and output it
0115 CDFA01            CALL    CONOUT          ;
0118 C1                POP     B               ;
0119 E1                POP     H               ; Next byte
011A 23                INX     H               ;
011B 05                DCR     B               ; End?
011C C21201            JNZ     MESLOOP         ; No - loop
011F 3EC3              MVI     A,(JMP)         ; Set up the values required for CP/M:
0121 320000            STA     0               ;   1. At 0000H - JMP Warm_Boot_entry
0124 210300            LXI     H,WBENTRY       ;
0127 220100            SHLD    1               ;
012A 320500            STA     5               ;   2. At 0005H - JMP BDOS_entry
012D 2106BD            LXI     H,BDOS          ;
0130 220600            SHLD    6               ;
0133 AF                XRA     A               ;   Set the current disk to A:
0134 320400            STA     CDISK           ;
0137 4F                MOV     C,A             ;
0138 CD3E01            CALL    AUTOLOAD        ; Gives the CCP the program to be
                                               ;    loaded on entry
013B C300B5            JMP     CCP             ; Start CP/M via CCP
               ;
               AUTOLOAD:
013E 2107B5            LXI     H,COMLINE       ; Command line space
0141 115201            LXI     D,PRONAME       ; Name of the program to be entered
0144 010C00            LXI     B,LENG          ;
               AUTOLOOP:
0147 1A                LDAX    D               ; Read the name
0148 77                MOV     M,A             ; Save it in the correct space
0149 23                INX     H               ; adjust the pointers
```

```
014A 13                 INX     D               ;
014B 0B                 DCX     B               ; End of the line?
014C 78                 MOV     A,B             ;
014D B1                 ORA     C               ;
014E C24701             JNZ     AUTOLOOP        ; No - then loop
0151 C9                 RET                     ;    or return
                ;
                PRONAME:
0152 0A413A4350         DB      0AH,'A:CPNETLDR',00H
000C =          LENG    EQU     $-PRONAME
                ;
015E 1B481B4A   MESSAGE DB      ESC,'H',ESC,'J' ; Clear the console
0162 2020202020         DB      '       UCT - CP/NET System',0DH,0AH
0021 =          MESLEN  EQU     $-MESSAGE
                ;
                ;-----------------------------------------------------------------------
                ;               THE WARM BOOT IS NOW UNDER NDOS CONTROL                 £
                ;-----------------------------------------------------------------------
                WBOOT:
017F 21FFFF             LXI     H,0FFFFH        ; As an error indicator
0182 7C                 MOV     A,H             ;
0183 C9                 RET
                page
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                        SECTION 4 - DISK PACKAGE                           *
                ;*                        ========================                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;-----------------------------------------------------------------------------£
                ;       Main Procedures                                                       £
                ;       ===============                                                      £
                ;                                                                             £
                ;       HOME    :       Set the constant address values to zero address       £
                ;       SELDSK  :       Check the local disk selection is A:                  £
                ;       SETTRK  :       Choose EPROM                                          £
                ;       SETSEC  :       Set the addresses for the 128 byte pages              £
                ;       SETDMA  :       Set dma address                                       £
                ;       READ    :       Read a block of data from the EPROM to memory         £
                ;       WRITE   :       Not avaliable                                         £
                ;       SECTRAN :       Has no effect                                         £
                ;                                                                             £
                ;-----------------------------------------------------------------------------
                ;
                ;
0023 =          EPNUM   EQU     23H             ; Mass storage card: EPROM number latch
0022 =          EPADRHI EQU     22H             ;                    Address byte latch (MSB)
0021 =          EPADRLO EQU     21H             ;                    Address byte latch (LSB)
0020 =          EPDATA  EQU     20H             ;                    Data register

                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION: HOME;                                                      £
                ;          Inputs:       None                                              £
                ;          Outputs:      None                                              £
                ;          Destroys:     A,H,L,flags                                       £
                ;          Calls:        None                                              £
                ;          Description:  Reads the Eprom offset and sets the Eprom number  £
                ;                        to this value.                                    £
                ;                                                                            £
                ;-----------------------------------------------------------------------------
                ;
                HOME:
0184 3A8B03             LDA     EPOSET          ; Read the EPROM offset value
0187 D323               OUT     EPNUM           ; Send it to the EPROM Number latch
0189 3E00               MVI     A,0             ;
018B D322               OUT     EPADRHI         ; Zero to the address latches
018D D321               OUT     EPADRLO         ;
018F C9                 RET                     ;
                        page
```

```
                ;----------------------------------------------------------------------------
                ;                                                                           £
                ;       FUNCTION SELDSK                                                     £
                ;          Inputs:      C: Selected disk number                             £
                ;          Outputs:     HL: base of disk parameter block                    £
                ;                       HL: Ø if an error occurred                          £
                ;          Destroys:    A,H,L,flags                                         £
                ;          Description: The disk number is set and, if valid, the base      £
                ;                       of the disk parameter block is returned in HL or    £
                ;                       if not, HL = Ø.                                     £
                ;                                                                           £
                ;----------------------------------------------------------------------------
                ;
                SELDSK:
Ø19Ø 79                 MOV     A,C             ;
Ø191 328903             STA     SEKDSK          ; Make a note of the requested disk
Ø194 FEØØ               CPI     Ø               ;
Ø196 C29DØ1             JNZ     ENDSELD         ; If it is not Ø (A:) then error
Ø199 213ØØØ             LXI     H,DSKHED        ; The base of the Disk Parameter Header
Ø19C C9                 RET                     ;
                ENDSELD:
Ø19D 21ØØØØ             LXI     H,Ø             ; H=Ø means error
ØlAØ C9                 RET                     ;

                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION: SETTRK                                                     £
                ;          Inputs:      BC: Selected track number                            £
                ;          Outputs:     None                                                 £
                ;          Destroys:    A                                                    £
                ;          Calls:       None                                                 £
                ;          Description: Set the EPROM number for later use.                  £
                ;                                                                            £
                ;-----------------------------------------------------------------------------
                ;
                SETTRK:
Ø1A1 E5                 PUSH    H               ; Destroy as little as possible
Ø1A2 3A8BØ3             LDA     EPOSET          ; Read the off set
Ø1A5 81                 ADD     C               ; The track number, = EPROM number
Ø1A6 328CØ3             STA     EPNUMS          ;
Ø1A9 D323               OUT     EPNUM           ; Set the EPROM number
Ø1AB E1                 POP     H               ;
Ø1AC C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------------£
                ;                                                                             £
                ;       FUNCTION: SETSEC                                                      £
                ;          Inputs:      BC: Selected sector number                            £
                ;          Outputs:     None                                                  £
                ;          Destroys:    H,L                                                   £
                ;          Calls:       None                                                  £
                ;          Description: Sets up the MSB value on the EPROM card and           £
                ;                       calculates the LSB address offset                     £
                ;                                                                             £
                ;-----------------------------------------------------------------------------£
                ;
                SETSEC:
01AD 79                 MOV     A,C             ; Read the sector number
01AE B7                 ORA     A               ; Clear the carry flag
01AF 1F                 RAR                     ; £128
01BØ 47                 MOV     B,A             ; B = Ø as max of 64 tracks
01B1 3EØØ               MVI     A,Ø             ; Get LSB offset
01B3 1F                 RAR                     ;
01B4 328AØ3             STA     ADROFF          ; Save the offset.
01B7 78                 MOV     A,B             ; Set the MSB of the EPROM
01B8 D322               OUT     EPADRHI         ;
01BA C9                 RET                     ;

                        page
```

```
                ;----------------------------------------------------------------------------
                ;                                                                           £
                ;       FUNCTION: SETDMA                                                    £
                ;          Inputs:      BC: Selected DMA address                            £
                ;          Outputs:     None                                                £
                ;          Destroys:    None                                                £
                ;          Calls:       None                                                £
                ;          Description: Set the selected DMA address.                       £
                ;                                                                           £
                ;----------------------------------------------------------------------------
                ;
                SETDMA:
01BB E5                 PUSH    H               ;
01BC 60                 MOV     H,B             ;
01BD 69                 MOV     L,C             ;
01BE 228D03             SHLD    DMAADR          ;
01C1 E1                 POP     H               ;
01C2 C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION READ                                                  £
                ;          Inputs:      None                                           £
                ;          Outputs:     A:=Ø => no error                               £
                ;                       A:=1 => error                                  £
                ;          Destroys:    All                                            £
                ;          Calls:       None                                           £
                ;          Description: Reads a 'Sector' of data from the EPROM card and puts £
                ;                       it into the buffer pointed to by the DMAADR pointer. £
                ;                       A zero in the A register means that there was no £
                ;                       error a 1 means an error occured.              £
                ;-----------------------------------------------------------------------
                ;
                READ:
Ø1C3 2A8DØ3             LHLD    DMAADR          ; Reads the address of the buffer
Ø1C6 EB                 XCHG                    ;
Ø1C7 ØEØØ               MVI     C,Ø             ; Counter
Ø1C9 3A8AØ3             LDA     ADROFF          ; Read the offset (using 128 byte pages)
Ø1CC 47                 MOV     B,A             ; Save it in the B register
Ø1CD 3EØØ               MVI     A,Ø             ;
                RL1:                            ;
Ø1CF BØ                 ORA     B               ; Include the offset bit
Ø1DØ D321               OUT     EPADRLO         ; Send it to the EPROM card
Ø1D2 DB2Ø               IN      EPDATA          ; Read the byte at that address
Ø1D4 12                 STAX    D               ; Store in the memory buffer
Ø1D5 13                 INX     D               ;
Ø1D6 ØC                 INR     C               ;
Ø1D7 79                 MOV     A,C             ; Check for 128
Ø1D8 FE8Ø               CPI     128             ;
Ø1DA DACFØ1             JC      RL1             ;
Ø1DD 3EØØ               MVI     A,Ø             ; No error
Ø1DF C9                 RET                     ;
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                             £
                ;       FUNCTION WRITE                                                        £
                ;          Inputs:                                                            £
                ;          Outputs:                                                           £
                ;          Destroys:    A                                                     £
                ;          Calls:       None                                                  £
                ;          Description: This routine should not be called and so there is an  £
                ;                       error return if it is called.                         £
                ;                                                                             £
                ;-----------------------------------------------------------------------------
                ;
                WRITE:
01E0 3E01               MVI     A,1             ; As an error indicator
01E2 C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION SECTRAN                                               £
                ;          Inputs:      BC:=sector to translate                        £
                ;          Outputs:     HL:=translated sector                          £
                ;          Destroys:    B,C,H,L                                        £
                ;          Calls:       None                                           £
                ;          Description: The sector translation is not  performed  until £
                ;                       deblocking and is therefore returned unchanged.£
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                SECTRAN:
01E3 60                 MOV     H,B             ; Simply duplicate the sector
01E4 69                 MOV     L,C             ;
01E5 C9                 RET                     ;

                        page
```

```
;*****************************************************************************
;*                                                                           *
;*                      SECTION 5 - Communications Package                   *
;*                      ==================================                   *
;*                                                                           *
;*****************************************************************************
;                                                                           £
;       Main Procedures                                                     £
;       ===============                                                     £
;                                                                           £
;       CONST   :       Console status                                      £
;       CONIN   :       Console character in                                £
;       CONOUT  :       Covsole character out                               £
;       LIST    :       List character out                                  £
;       PUNCH   :       Punch character out                                 £
;       READER  :       Reader character in                                 £
;       LISTST  :       List status                                         £
;                                                                           £
;
;       Auxiliary Procedures                                                £
;       ====================                                                £
;                                                                           £
;       BOOTCOM :       Initialise communications ports and protocols       £
;       SELDEV  :       Select hardware device                              £
;       RDRST   :       Reader status to transmit                           £
;       SRSERØ  :       Status serial-Ø port to receive                     £
;       STSERØ  :       Status serial-Ø port to transmit                    £
;       SRSERØ  :       Status serial-1 port to receive                     £
;       STSER1  :       Status serial-1 port to transmit                    £
;       STPARL  :       Status parallel port to transmit                    £
;       INSERØ  :       Input serial-Ø port                                 £
;       INSER1  :       Input serial-1 port                                 £
;       OUTSERØ :       Output serial-Ø port                                £
;       OUTSER1 :       Output serial-1 port                                £
```

```
                ;       OUTPARL :       Output parallel port                                    £
                ;       S1$ETX  :       Serial-1 ETX/ACK protocol transmit                      £
                ;       S1$XON  :       Serial-1 X-ON/X-OFF protocol transmit                   £
                ;       S1S$ETX :       Serial-1 ETX/ACK protocol status                        £
                ;       S1S$XON :       Serial-1 X-ON/X-OFF protocol status                     £
                ;                                                                               £
                ;-------------------------------------------------------------------------------
                ;
0000 =          COMCARD EQU     0H              ; Base address for the Communications
0000 =          SER0$D  EQU     COMCARD+0       ; Console port  - Data register
0001 =          SER0$C  EQU     COMCARD+1       ;                 - Command register
0004 =          SER1$D  EQU     COMCARD+4       ; Serial Port 1 - Data register
0005 =          SER1$C  EQU     COMCARD+5       ;                 - Command register
000C =          PARL$A  EQU     COMCARD+12      ; Parallel Port - Port A
000D =          PARL$B  EQU     COMCARD+13      ;                 - Port B
000E =          PARL$C  EQU     COMCARD+14      ;                 - Port C
000F =          PARL$M  EQU     COMCARD+15      ;                 - Command register
0002 =          RXRDY   EQU     00000010B       ; Receive Ready bit mask
0001 =          TXRDY   EQU     00000001B       ; Serial Transmit ready bit mask
0002 =          PTXRDY  EQU     00000010B       ; Parallel Transmit ready mask
007F =          NOPAR   EQU     7FH             ; Parity strip mask
0003 =          IOBYTE  EQU     0003H           ; The IO Byte address

                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION CONST                                                       £
                ;          Inputs:      None                                                 £
                ;          Outputs:     A:=ØØ => ready                                       £
                ;                       A:=FF => not ready                                   £
                ;          Destroys:    A,D,E,F,H,L                                          £
                ;          Calls:       SELDEV                                               £
                ;          Description: Return status of console device in "A" register.     £
                ;                                                                            £
                ;-----------------------------------------------------------------------------
                ;
                CONST:
Ø1E6 213202             LXI     H,@CONST        ; Base address of the Jump table
Ø1E9 3A0300             LDA     IOBYTE          ; Select the device
Ø1EC Ø7                 RLC                     ; £2
Ø1ED C37202             JMP     SELDEV          ; Jump to the jump table calculation
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION CONIN                                                 £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: character from console                      £
                ;          Destroys:    A,D,E,F,H,L                                    £
                ;          Calls:       SELDEV                                         £
                ;          Description: Waits until a character is ready and then returns £
                ;                       it in the "A" register.                        £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                CONIN:
01F0 213A02             LXI     H,@CONIN        ; Base address for the consoles
01F3 3A0300             LDA     IOBYTE          ; Adjust using the IO BYTE
01F6 07                 RLC                     ; £2
01F7 C37202             JMP     SELDEV          ;  and jump
                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION: CONOUT;                                                    £
                ;          Inputs:      C: character to send                                 £
                ;       Destroys:    A,D,E,F,H,L                                          £
                ;          Calls:       SELDEV                                            £
                ;          Description: Output a character to the console device.         £
                ;                                                                            £
                ;-----------------------------------------------------------------------------
                ;
                CONOUT:
01FA 214202             LXI     H,@CONOUT       ; Console output base address
01FD 3A0300             LDA     IOBYTE          ; Adjust the address using the IO Byte
0200 07                 RLC                     ; £2
0201 C37202             JMP     SELDEV          ;

                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION: LIST                                                 £
                ;          Inputs:      C: character to list                           £
                ;          Outputs:     None                                           £
                ;          Destroys:    A,D,E,F,H,L                                    £
                ;          Calls:       SELDEV                                         £
                ;          Description: Output a character to the listing device.      £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                LIST:
0204 214A02             LXI     H,@LIST         ; Base address for the List devices
0207 3A0300             LDA     IOBYTE          ; Adjust using the IO Byte
020A 07                 RLC                     ; /32
020B 07                 RLC                     ;
020C 07                 RLC                     ;
020D C37202             JMP     SELDEV          ;
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION: PUNCH                                                £
                ;          Inputs:      C: character to punch                          £
                ;          Outputs:     None                                           £
                ;          Destroys:    A,D,E,F,H,L                                    £
                ;          Calls:       SELDEV                                         £
                ;          Description: Output a character to the punch device.        £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                PUNCH:
0210 215202             LXI     H,@PUNCH        ; Base address for the Punch devices
0213 3A0300             LDA     IOBYTE          ; Using the IO Byte / 8
0216 0F                 RRC                     ;
0217 0F                 RRC                     ;
0218 0F                 RRC                     ;
0219 C37202             JMP     SELDEV          ;
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION READER                                                £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: character from punch device                 £
                ;          Destroys:    A,D,E,F,H,L                                    £
                ;          Calls:       SELDEV                                         £
                ;          Description: Input the character from the reader device.    £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                READER:
021C 215A02             LXI     H,@READER       ; Base address of the Reader group
021F 3A0300             LDA     IOBYTE          ; Use the IO Byte once more
0222 0F                 RRC                     ; /2
0223 C37202             JMP     SELDEV          ;
                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                             £
                ;       FUNCTION LISTST                                                       £
                ;          Inputs:      None                                                  £
                ;          Outputs:     A: ØØ => ready                                        £
                ;                       A: FF => not ready                                    £
                ;          Destroys:    A,D,E,F,H,L                                           £
                ;          Calls:       SELDEV                                                £
                ;          Description: Return the status of the listing device.              £
                ;                                                                             £
                ;-----------------------------------------------------------------------------
                ;
                LISTST:
Ø226 216202             LXI     H,@LISTST       ; Read the Base address for the List devices
Ø229 3AØ3ØØ             LDA     IOBYTE          ; Using the IO Byte calculate the jump address
Ø22C Ø7                 RLC                     ;
Ø22D Ø7                 RLC                     ;
Ø22E Ø7                 RLC                     ;
Ø22F C37202             JMP     SELDEV          ;
                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION: SELDEV                                                     £
                ;          Inputs:      A:=number ....  *2 for address width                 £
                ;                       H:=group                                           £
                ;          Outputs:     None                                               £
                ;          Destroys:    A,D,E,F,H,L                                        £
                ;          Calls:       SRSER0,SRSER1,STSER0,STSER1,STPARL,                £
                ;                       RDRST,READER,LIST,                                 £
                ;                       INSER0,INSER1,OUTSER0,OUTSER1,OUTPARL,             £
                ;                       S1S$ETX,S1S$XON,S1$ETX,S1$XON                      £
                ;          Description: According to the state of the "IOBYTE", the logical £
                ;                       device is assigned to the physical device.         £
                ;                                                                          £
                ;-----------------------------------------------------------------------------
                ;
                @CONST:                         ; Jump addresses for the different routines
0232 AB02               DW      SRSER0          ;
0234 B302               DW      SRSER1          ;
0236 7D02               DW      RDRST           ;
0238 B302               DW      SRSER1          ;
                @CONIN:
023A D302               DW      INSER0          ;
023C DF02               DW      INSER1          ;
023E 1C02               DW      READER          ;
0240 DF02               DW      INSER1          ;
                @CONOUT:
0242 EB02               DW      OUTSER0         ;
0244 F602               DW      OUTSER1         ;
0246 0402               DW      LIST            ;
0248 F602               DW      OUTSER1         ;
                @LIST:
024A EB02               DW      OUTSER0         ;
024C 1603               DW      S1$ETX          ;
```

```
024E 2403               DW      S1$XON          ;
0250 0103               DW      OUTPARL         ;
                @PUNCH:
0252 EB02               DW      OUTSER0         ;
0254 F602               DW      OUTSER1         ;
0256 F602               DW      OUTSER1         ;
0258 F602               DW      OUTSER1         ;
                @READER:
025A D302               DW      INSER0          ;
025C DF02               DW      INSER1          ;
025E DF02               DW      INSER1          ;
0260 DF02               DW      INSER1          ;
                @LISTST:
0262 BB02               DW      STSER0          ;
0264 2E03               DW      S1S$ETX         ;
0266 5F03               DW      S1S$XON         ;
0268 CB02               DW      STPARL          ;
                @RDRST:
026A AB02               DW      SRSER0          ;
026C B302               DW      SRSER1          ;
026E B302               DW      SRSER1          ;
0270 B302               DW      SRSER1          ;
                ;
                SELDEV:
0272 E606               ANI     110B            ; Calculate the Jump address from the
0274 5F                 MOV     E,A             ;  input
0275 1600               MVI     D,0             ;
0277 19                 DAD     D               ;
0278 7E                 MOV     A,M             ;
0279 23                 INX     H               ;
027A 66                 MOV     H,M             ;
027B 6F                 MOV     L,A             ;
027C E9                 PCHL                    ; Continue
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION RDRST                                                 £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: ØØ => ready                                 £
                ;                       A: FF => not ready                             £
                ;          Destroys:    A,D,E,F,H,L                                    £
                ;          Calls:       SELDEV                                         £
                ;          Description: Return the status of the reader device.        £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                RDRST:
027D 216A02             LXI     H,@RDRST        ; If the Card Reader is the console device
0280 3A0300             LDA     IOBYTE          ;   calculate the jump again
0283 0F                 RRC
0284 C37202             JMP     SELDEV          ;

                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION: BOOTCOM                                              £
                ;          Inputs:      None                                           £
                ;          Outputs:     None                                           £
                ;          Destroys:    A                                              £
                ;          Calls:       None                                           £
                ;          Description: Initialise the serial and parallel communications £
                ;                       ports as well as the IOBYTE.                   £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
0040 =          RSET51  EQU     040H            ; Reset value
00CE =          MODE51  EQU     0CEH            ; 2 stop bits,no parity,8 data bits,async X16
0037 =          CMD51   EQU     037H            ; RST,error reset,receive enable,
                                                ;   terminal ready,transmit enable
0081 =          MODE55  EQU     081H            ; Mode set active,
                                                ;       A: mode 0
                                                ;          output
                                                ;          C hi output
                                                ;       B: mode 0
                                                ;          ouput
                                                ;          C low input
0094 =          INTIOBY EQU     094H            ; The initial IO Byte:  CON:=TTY:
                                                ;                       RDR:=PTR:
                                                ;                       PUN:=PTP:
                                                ;                       LST:=LPT:
00FF =          NODATA  EQU     0FFH            ; If there is no data
                ;
                BOOTCOM:
0287 3E40               MVI     A,RSET51        ; Reset the USART
0289 D301               OUT     SER0$C          ;
028B D305               OUT     SER1$C          ;
028D 3ECE               MVI     A,MODE51        ; Set up the mode
```

```
028F D301           OUT     SER0$C          ;
0291 D305           OUT     SER1$C          ;
0293 3E37           MVI     A,CMD51         ; And then the Command mode
0295 D301           OUT     SER0$C          ;
0297 D305           OUT     SER1$C          ;
0299 DB00           IN      SER0$D          ; Read it
029B DB04           IN      SER1$D          ;
029D 3E81           MVI     A,MODE55        ; Set up the Parallel port
029F D30F           OUT     PARL$M          ;
02A1 3EFF           MVI     A,NODATA        ;
02A3 D30D           OUT     PARL$B          ;
                ;
02A5 3E94           MVI     A,INTIOBY       ; Set the IO Byte to the initial value
02A7 320300         STA     IOBYTE          ;
02AA C9             RET                     ;

                    page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                             £
                ;       FUNCTION SRSERØ                                                       £
                ;          Inputs:      None                                                  £
                ;          Outputs:     A: FF => ready                                        £
                ;                       A: ØØ => not ready                                    £
                ;          Destroys:    A,F                                                   £
                ;          Calls:       None                                                  £
                ;          Description: Returns the receiving status of the serial-A port.    £
                ;                                                                             £
                ;-----------------------------------------------------------------------------
                ;
                SRSERØ:
02AB DBØ1               IN      SERØ$C          ; Read the status
02AD E6Ø2               ANI     RXRDY           ; Ready?
02AF C8                 RZ                      ;   No - Return with A = Ø
02BØ F6FF               ORI     ØFFH            ;   Yes - Return with A = FF
02B2 C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION SRSER1                                                      £
                ;          Inputs:      None                                                 £
                ;          Outputs:     A: FF => ready                                       £
                ;                       A: ØØ => not ready                                   £
                ;          Destroys:    A,F                                                  £
                ;          Calls:       None                                                 £
                ;          Description: Returns the receiving status of the serial-B port.   £
                ;                                                                            £
                ;-----------------------------------------------------------------------------
                ;
                SRSER1:
02B3 DB05               IN      SER1$C          ; Read the Status
02B5 E602               ANI     RXRDY           ; Ready?
02B7 C8                 RZ                      ;  no - set A = Ø and return
02B8 F6FF               ORI     ØFFH            ;  yes - set A = FF and return
02BA C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION STSERØ                                                      £
                ;          Inputs:      None                                                 £
                ;          Outputs:     A:=FF => ready                                       £
                ;                       A:=ØØ => not ready                                   £
                ;          Destroys:    A,F                                                  £
                ;          Calls:       None                                                 £
                ;          Description: Returns the transmit status of the serial-A port.    £
                ;                                                                            £
                ;-----------------------------------------------------------------------------
                STSERØ:
02BB DB01               IN      SERØ$C          ; Read the status
02BD E601               ANI     TXRDY           ; Ready to transmit?
02BF C8                 RZ                      ;  no - return with A = Ø
02C0 F6FF               ORI     ØFFH            ;  yes - return with A = FF
02C2 C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION STSER1                                                £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: FF => ready                                 £
                ;                       A: Ø => not ready                              £
                ;          Destroys:    A,F                                            £
                ;          Calls:       None                                           £
                ;          Description: Returns the transmit status of the serial-B port. £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                STSER1:
Ø2C3 DBØ5               IN      SER1$C          ; Read the Status
Ø2C5 E6Ø1               ANI     TXRDY           ; Ready to Send?
Ø2C7 C8                 RZ                      ;   no - Return with A = Ø
Ø2C8 F6FF               ORI     ØFFH            ;   yes - Return with A = FF
Ø2CA C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION STPARL                                                £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: FF => ready                                 £
                ;                       A: ØØ => not ready                             £
                ;          Destroys:    A,F                                            £
                ;          Calls:       None                                           £
                ;          Description: Returns the transmit status of the parallel port.  £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                STPARL:
ØØ2CB DBØE              IN      PARL$C          ; Read the parallel port status
ØØ2CD E6Ø2              ANI     PTXRDY          ; Ready to send?
ØØ2CF 3D                DCR     A               ;   yes - set A =ØFFH and return
ØØ2DØ F8                RM                      ;
ØØ2Dl AF                XRA     A               ;   no - set A =ØØH and return
ØØ2D2 C9                RET                     ;

                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION INSERØ                                                £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: character                                   £
                ;          Destroys:    A,F                                            £
                ;          Calls:       None                                           £
                ;          Description: Waits until a character is ready and returns it £
                ;                       from the serial-A port.                        £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                INSERØ:
02D3 DB01               IN      SERØ$C          ; Read the Status
02D5 E602               ANI     RXRDY           ;  is it ready to transmit
02D7 CAD302             JZ      INSERØ          ;  no - loop
02DA DB00               IN      SERØ$D          ;  yes - Read the input register
02DC E67F               ANI     NOPAR           ;     and remove the parity
02DE C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------------
                ;                                                                            £
                ;       FUNCTION INSER1                                                      £
                ;          Inputs:      None                                                 £
                ;          Outputs:     A:=character                                         £
                ;          Destroys:    A,F                                                  £
                ;          Calls:       None                                                 £
                ;          Description: Waits until a character is ready and returns it      £
                ;                       from the serial-B port.                              £
                ;                                                                            £
                ;-----------------------------------------------------------------------------
                ;
                INSER1:
02DF DB05               IN      SER1$C          ; Read the status register
02E1 E602               ANI     RXRDY           ;   and wait until it is ready
02E3 CADF02             JZ      INSER1          ;
02E6 DB04               IN      SER1$D          ;   then read the character and
02E8 E67F               ANI     NOPAR           ;       remove its parity bit.
02EA C9                 RET                     ;

                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION: OUTSERØ                                              £
                ;          Inputs:      C: character                                   £
                ;          Outputs:     None                                           £
                ;          Destroys:    A,F                                            £
                ;          Calls:       None                                           £
                ;          Description: Outputs a character to the serial-A port when  £
                ;                       it is ready.                                   £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                OUTSERØ:
02EB DB01               IN      SERØ$C          ; Read the Status register
02ED E601               ANI     TXRDY           ;    and wait until it is ready to send
02EF CAEB02             JZ      OUTSERØ         ;
02F2 79                 MOV     A,C             ; Then send the character
02F3 D300               OUT     SERØ$D          ;
02F5 C9                 RET                     ;
                        page
```

```
                ;----------------------------------------------------------------------------
                ;                                                                           £
                ;       FUNCTION: OUTSER1                                                   £
                ;          Inputs:      C: character                                        £
                ;          Outputs:     None                                                £
                ;          Destroys:    A,F                                                 £
                ;          Calls:       None                                                £
                ;          Description: Outputs a character to the serial-B port when       £
                ;                       it is ready.                                        £
                ;                                                                           £
                ;----------------------------------------------------------------------------
                ;
                OUTSER1:
02F6 DB05               IN      SER1$C          ; Read the status register
02F8 E601               ANI     TXRDY           ;   and wait until it is ready to send
02FA CAF602             JZ      OUTSER1         ;
02FD 79                 MOV     A,C             ;  then send the character
02FE D304               OUT     SER1$D          ;
0300 C9                 RET                     ;

                        page
```

```
                ;----------------------------------------------------------------------------
                ;                                                                           £
                ;       FUNCTION: OUTPARL                                                   £
                ;          Inputs:      C: character                                        £
                ;          Outputs:     None                                                £
                ;          Destroys:    A,F                                                 £
                ;          Calls:       None                                                £
                ;          Description: Outputs a character to the parallel port when       £
                ;                       it is ready. # CENTRONICS FORMAT #                  £
                ;                                                                           £
                ;----------------------------------------------------------------------------
                ;
0080 =          NOSTB   EQU     10000000B       ; No strobe bit
007F =          STROBE  EQU     01111111B       ; The strobe character
                ;
                OUTPARL:
0301 DB0E               IN      PARL$C          ; Read the status register
0303 E602               ANI     PTXRDY          ; Wait until the transmit bit is set
0305 C20103             JNZ     OUTPARL         ;
0308 3E80               MVI     A,NOSTB         ; Send the character to the port
030A B1                 ORA     C               ;  with no strobe initially
030B D30D               OUT     PARL$B          ;
030D E67F               ANI     STROBE          ; And then with the strobe
030F D30D               OUT     PARL$B          ;
0311 F680               ORI     NOSTB           ; Reset the strobe line
0313 D30D               OUT     PARL$B          ;
0315 C9                 RET                     ;
                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION: S1$ETX                                               £
                ;          Inputs:      C: character                                   £
                ;          Outputs:     None                                           £
                ;          Destroys:    A,F,H,L                                        £
                ;          Calls:       S1S$ETX                                        £
                ;          Description: Output a character to the serial-B port according £
                ;                       to the ETX/ACK protocol. # DIABLO PRINTER #    £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                S1$ETX:
0316 CD2E03             CALL    S1S$ETX         ; Get the status of Port B using ETX/ACK
0319 CA1603             JZ      S1$ETX          ; Wait until ready
031C 79                 MOV     A,C             ;    then send it
031D D304               OUT     SER1$D          ;
031F 218803             LXI     H,ETX$CNT       ; Counter-1
0322 35                 DCR     M               ;
0323 C9                 RET                     ;

                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION: S1$XON                                               £
                ;          Inputs:      C: character                                   £
                ;          Outputs:     None                                           £
                ;          Destroys:    A,F,H,L                                        £
                ;          Calls:       S1S$XON                                        £
                ;          Description: A character is sent to the serial-B port when it £
                ;                       is ready according to the X-OFF/X-ON protocol. £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
                S1$XON:
0324 CD5F03             CALL    S1S$XON         ; Wait until the port is ready to
0327 CA2403             JZ      S1$XON          ;   using the X-ON/X-OFF method
032A 79                 MOV     A,C             ;   then transmit the character
032B D304               OUT     SER1$D          ;
032D C9                 RET                     ;

                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION S1S$ETX                                               £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: FF => ready                                 £
                ;                       A: ØØ => not ready                             £
                ;          Destroys:    A,F,H,L                                        £
                ;          Calls:       None                                           £
                ;          Description: The status of the serial-B port is returned    £
                ;                       according to the ETX/ACK protocol.             £
                ;                       The values of ETXCNT: Ø = Send ETX if ready    £
                ;                                             FF = Is port ready       £
                ;                                             5Ø = Receive ACK if ready £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
0003 =          ETX     EQU     Ø3H             ; ASCII value
0006 =          ACK     EQU     Ø6H             ;
                ;
                S1S$ETX:
032E 218803             LXI     H,ETXCNT        ;
0331 AF                 XRA     A               ;
0332 BE                 CMP     M               ;
0333 C24203             JNZ     CASE1           ; If it is zero then send ETX
0336 DB05               IN      SER1$C          ;   if the port is ready
0338 E601               ANI     TXRDY           ;
033A C8                 RZ                      ; Return with A = Ø means not ready
033B 35                 DCR     M               ; Change the count to ETX sent
033C 3E03               MVI     A,ETX           ;   and send the ETX
033E D304               OUT     SER1$D          ;
0340 AF                 XRA     A               ; Set A=Ø as not yet ready
0341 C9                 RET                     ;
                CASE1:                          ;
0342 F25703             JP      CASELSE         ;
```

```
0345 DB05               IN      SER1$C          ; ACK received?
0347 E602               ANI     RXRDY           ;
0349 C8                 RZ                      ; Nothing at the port, return with A = 0
034A DB04               IN      SER1$D          ; Is the character an ACK?
034C E67F               ANI     NOPAR           ;
034E FE06               CPI     ACK             ;
0350 CA5503             JZ      ACKREC          ; yes - Adjust counter and see if ready
                                                ;       to transmit
0353 AF                 XRA     A               ; no  - Set A = 0 and return
0354 C9                 RET                     ;
                ACKREC:
0355 3632               MVI     M,50            ; To show that ACK has been received
                ;
                CASELSE:
0357 DB05               IN      SER1$C          ; Read the status register
0359 E601               ANI     TXRDY           ; Ready to send?
035B C8                 RZ                      ; no - Set A = 0 and return
035C F6FF               ORI     0FFH            ; yes - Set A = 0FFH and Return
035E C9                 RET                     ;

                        page
```

```
                ;-----------------------------------------------------------------------
                ;                                                                      £
                ;       FUNCTION S1S$XON                                               £
                ;          Inputs:      None                                           £
                ;          Outputs:     A: FF => ready                                 £
                ;                       A: ØØ => not ready                             £
                ;          Destroys:    A,F,H,L                                        £
                ;          Calls:       None                                           £
                ;          Description: The status of the serial-B port is returned    £
                ;                       according to the X-ON/X-OFF protocol.          £
                ;                                                                      £
                ;-----------------------------------------------------------------------
                ;
0011 =          XON     EQU     11H             ; ASCII values
0013 =          XOFF    EQU     13H             ;
                ;
                S1S$XON:
035F 218703             LXI     H,XONFLAG       ; Set pointer to the flag
0362 DB05               IN      SER1$C          ; Read the status
0364 E602               ANI     RXRDY           ; Receive ready?
0366 CA7E03             JZ      NOINPUT         ; no - Try for output ready
0369 DB04               IN      SER1$D          ; yes - Read the data
036B E67F               ANI     NOPAR           ;   removing the parity bit
036D FE11               CPI     XON             ; Is it an X-ON character?
036F C27703             JNZ     CNTL$S          ; no - Try for X-OFF
0372 36FF               MVI     M,0FFH          ; yes - Set flag
0374 C37E03             JMP     NOINPUT         ;
                CNTL$S:
0377 FE13               CPI     XOFF            ; Is it a X-OFF?
0379 C27E03             JNZ     NOINPUT         ; no - then ignore it
037C 3600               MVI     M,0             ; yes - Reset the flag
                ;                               ;
                NOINPUT:
037E DB05               IN      SER1$C          ; Read the Status byte
```

```
0380 E601           ANI     TXRDY           ; Is the Ready bit clear
0382 A6             ANA     M               ;  or the Flag = 0?
0383 C8             RZ                      ; yes - Return with A=0
0384 F6FF           ORI     0FFH            ; no - Set A=0FFH and return
0386 C9             RET                     ;

                    page
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*              SECTION 6 - Variables of the Communications Package *
                ;*              ==================================================== *
                ;*                                                               *
                ;*****************************************************************
                ;
0387 FF         XONFLAG DB      0FFH            ; FF=X-ON received
                                                ; 0=X-OFF received
0388 32         ETXCNT  DB      50              ; 0=Transmit ETX
                                                ; 50=ACK received
                                                ; FF=Wait for Port to ready
                ;
                ;*****************************************************************
                ;*                                                               *
                ;*              SECTION 7 - Variables of the Disk Package and DPBlock *
                ;*              ==================================================== *
                ;*                                                               *
                ;*****************************************************************
                ;
0389 00         SEKDSK  DB      0               ; Selected disk
038A            ADROFF  DS      1               ; Gives the address offset for 128 byte pages
038B            EPOSET  DS      1               ; Saves the first EPROM number
038C            EPNUMS  DS      1               ; Saves the number of the present EPROM
                ;
038D 8000       DMAADR  DW      0080H           ; DMA address storage space
                ;
038F                    END
```

APPENDIX B

The Cold Start Loader for the UCT Micro-computers

```
                ;==========================================================================
                ;
                ;                               EQUATES
                ;
                ;==========================================================================
ØDØA =          LF      EQU     ØDØAH           ; Caraige return line feed
1ØØØ =          STRT    EQU     1ØØØH           ; The start address
F3ØØ =          CPMBOOT EQU     ØF3ØØH          ; The entry point for BIOS
DDØØ =          NEWPOS  EQU     ØDDØØH          ; Start of BDOS
1A8Ø =          LEN     EQU     1A8ØH           ; Length of the BDOS, BIOS and CCP
ØØ8Ø =          SAVSP   EQU     Ø8ØH            ; Save space for transfer of data
ØØ23 =          ENUM    EQU     23H             ; Eprom Number address
ØØ22 =          EADHI   EQU     22H             ; High Address on the Eprom selected
ØØ21 =          EADLO   EQU     21H             ; Low Address on the Eprom selected
ØØ2Ø =          EDATA   EQU     2ØH             ; Data from the Eprom selected
ØØ1Ø =          MAXEP   EQU     16              ; Maximum number of EPROMs on the
                                                ;    Mass storage card
F3D =           OUTMSG  EQU     ØF3DH           ; SABus string output routine
                page
```

Carriage

```
                ;==========================================================================
                ;
                ;                           LOCAL DATA SEGMENT
                ;
                ;==========================================================================

                        DSEG
                EPNUM:
0000                    DS      1               ; Save the present EPROM number
                SAVED:
0001                    DS      2               ; Save space for CCP address
0003                    DS      50              ; Stack space
                STAC:
                        page
```

```
                ;=================================================================
                ;
                ;               START OF MAIN PROGRAM
                ;
                ;=================================================================
                ;
                        CSEG

0000 CD3D00             CALL    WHICHEP         ; Find the start of the 'Eprom disk'
0003 3C                 INR     A               ; An error return?
0004 CA1D00             JZ      NODIR           ;
0007 3D                 DCR     A               ;
0008 320000             STA     EPNUM           ; Save the current eprom number
000B 313500             LXI     SP,STAC         ; Set up the stack pointer
000E 116600             LXI     D,OLDPOS        ; DE = Position of BIOS and BDOS
0011 0100DD             LXI     B,NEWPOS        ; BC = The new position
0014 21801A             LXI     H,LEN           ; HL = Length of the data
0017 CD5B00             CALL    TXLOOP          ; Move the data
001A C300F3             JMP     CPMBOOT         ; Boot up using the BIOS

                NODIR:
001D 212400             LXI     H,ERRMSG        ; Send an error message
0020 CD3DF0             CALL    OUTMSG          ;
0023 76                 HLT
                ;
                ERRMSG:
0024 0A0D07074E         DB      0AH,0DH,07H,07H,'No Directory found',0DH,0AH,00H
                        page
```

```
                ;
                ;--------------------------------------------------------------------------
                ; FUNCTION: WHICHEP
                ; INPUTS:   None
                ; OUTPUTS:  A - Start of the Eprom Disk less one
                ; CALLS:    None
                ; DESCRIPTION:
                ;       WHICHEP reads the first byte on each EPROM on the mass storage card
                ;       until it finds a 00H. This must be the start of the directory and so
                ;       it returns the EPROM number.
                ;
                ;--------------------------------------------------------------------------
                ;
                WHICHEP:
003D 0E00               MVI     C,0             ; Try the first EPROM
003F 79                 MOV     A,C
                TRYNEXT:
0040 D323               OUT     ENUM            ;
0042 3E00               MVI     A,0             ;
0044 D322               OUT     EADHI           ; And set the addresses to zero
0046 D321               OUT     EADLO           ;
0048 DB20               IN      EDATA           ; Read the data
004A FE00               CPI     0               ; Is it zero
004C CA5900             JZ      FOUND           ; Goodie
004F 0C                 INR     C               ; End of the EPROMs ?
0050 79                 MOV     A,C             ;
0051 FE10               CPI     MAXEP           ;
0053 C24000             JNZ     TRYNEXT         ;
0056 3EFF               MVI     A,0FFH          ; Error return
0058 C9                 RET                     ;
                FOUND:
0059 79                 MOV     A,C             ;
005A C9                 RET                     ;
```

```
                ;-----------------------------------------------------------------------
                ; FUNCTION: TXLOOP
                ; INPUTS:   BC - points at the new position
                ;           DE - containing the pointer pointing at the data
                ;           HL - Contains the length to be transferred
                ;           Data in the old position
                ; OUTPUTS:  Data to the new position
                ; CALLS:    None
                ; DESCRIPTION:
                ;           This routine reads the data from the old position in memory and
                ;           puts it in the new data position.
                ;
                ;-----------------------------------------------------------------------
                ;
                TXLOOP:
005B 1A                 LDAX    D               ; Read the data from the old position
005C 02                 STAX    B               ; Put it in the new position
005D 13                 INX     D               ; Point at the new address
005E 03                 INX     B               ;
005F 2B                 DCX     H               ; Check the length
0060 7C                 MOV     A,H             ;
0061 B5                 ORA     L               ;
0062 C25B00             JNZ     TXLOOP          ;
0065 C9                 RET                     ; Finished
                ;
                OLDPOS:
                        ;****************INCLUDE CCP, BDOS AND BIOS*******************

0066                    END
```

APPENDIX C

The EPROM Management System

EPROM Management System

```
(* 15/Ø1/85 *)
(*****************************************************************************)
(*                                                                           *)
(*  EPROM Management System                                                  *)
(*  =======================                                                  *)
(*                                                                           *)
(*  This program has a number of sections to it:                             *)
(*      1. EPROM map display                                                 *)
(*      2. File addition                                                     *)
(*      3. File deletion                                                     *)
(*                                                                           *)
(*  1. The EPROM display is a diagrammatic display of the programs as they   *)
(*     are found on the EPROM Card in the UCT SABus kits.                    *)
(*                                                                           *)
(*  2. File addition enables the user to add a file to the EPROM card and    *)
(*     then tells the user which EPROMs must be reprogrammed.                *)
(*                                                                           *)
(*  3. File deletion lets the user take a file off the mass storage card     *)
(*     and so make space for other programs.                                 *)
(*                                                                           *)
(*****************************************************************************)
PROGRAM EMS;

CONST   CPM_BLK = 128;
        MAXENT  = 128;

TYPE    DRIVES  = $Ø..$ØF;
        FCB     = RECORD
                      DRIVE : DRIVES;
                      FILENAME : PACKED ARRAY[1..11] OF CHAR;
                      SCRATCH1,SCRATCH2,SCRATCH3 : BYTE;
                      NUM_BLKS : BYTE;
                      BLOC_LOC : ARRAY[1..16] OF BYTE;
                  END;
```

```
        FYLE = FILE OF BYTE;
        FYL  = FILE OF FCB;

VAR     INFILE  : FYL;
        BUF     : ARRAY[1..MAXENT] OF FCB;
        BUFR    : ARRAY[1..MAXENT] OF BYTE;
        PLOT    : ARRAY[1..MAXENT] OF BOOLEAN;
        OFFSET,RESULT,MAXENTRY : INTEGER;
        ERROR   : BOOLEAN;
        ORDER   : ARRAY[1..MAXENT] OF INTEGER;
        N : FYLE;
        E : FYLE;
        ANS,NEW_FILE,EP_FILE : STRING;
        RESPONSE : CHAR;
```

```
(*************************************************************************)
(*                      BEGIN THE GENERAL UTILITY PROCEDURES             *)
(*************************************************************************)

EXTERNAL PROCEDURE @HLT;

(*-----------------------------------------------------------------------*)

FUNCTION ERRCHK : BOOLEAN;
(* This procedure checks to see if there were any errors in the SEEKREAD call*)
VAR
    ERR : BOOLEAN;

BEGIN
    ERROR := FALSE;
    IF IORESULT <> 0 THEN
        ERROR := TRUE;
    ERRCHK := ERR;
END;

(*-----------------------------------------------------------------------*)
```

```
EPROM Management System

FUNCTION READ_DIR(VAR DIR_FYL : FYL) : INTEGER;
(* Reads the Records from the EPDIR file to a buffer *)
VAR
    I     : INTEGER;
    ERROR : BOOLEAN;

BEGIN
    I := Ø;
    REPEAT
        SEEKREAD(DIR_FYL,I);
        ERROR := ERRCHK;
        I := I+1;
        IF I < MAXENT THEN
        BEGIN
            IF NOT ERROR THEN
                BUF[I] := DIR_FYL;
        END
        ELSE
            ERROR := TRUE;
    UNTIL ERROR;
    I := Ø;
    REPEAT
        I := I+1;
    UNTIL BUF[I].DRIVE = $E5;
    READ_DIR := (I-1);
END;

(*-------------------------------------------------------------------------*)
```

```
PROCEDURE SORT_ENTRIES(TOP_ENT : INTEGER);
(* Sorts the entries in an array pointer so that the records will be in *)
(*   order when printed.*)
VAR     I,J,TEMP : INTEGER;
        X        : FCB;
BEGIN
    FOR I := 1 TO TOP_ENT DO
    BEGIN
        FOR J := I+1 TO TOP_ENT DO
        BEGIN
            IF BUF[I].BLOC_LOC[1] > BUF[J].BLOC_LOC[1] THEN
            BEGIN
                X := BUF[J];
                BUF[J] := BUF[I];
                BUF[I] := X;
            END;
        END;
    END;
END;
```

```
(*****************************************************************************)
(*              UTILITY PROCEDURES FOR CONSOLE DISPLAY                       *)
(*****************************************************************************)

PROCEDURE SET_UP_CON;
(* Outputs the basic headings to the console.*)
BEGIN
    WRITE('£----------£-----------------------------');
    WRITELN('-------------------------£');
    WRITE('£  EPROM   £                          2k Block Number');
    WRITELN('                    £');
    WRITE('£  NUMBER  £      Ø       £      1      £');
    WRITELN('      2      £      3      £');
    WRITE('£----------£-------------£-------------£');
    WRITELN('-------------£-------------£');
    WRITE('£     Ø    £ ');
END;

(*---------------------------------------------------------------------------*)
```

```
PROCEDURE FILL_BLOCK(X,Y : INTEGER);
(* Fills the empty blocks*)
VAR     I : INTEGER;
BEGIN
    IF X < 64 THEN
    BEGIN
        WRITE('    ---     £ ');
        FOR I := (X+1) TO Y DO
        BEGIN
            IF I MOD 4 =Ø THEN
                BEGIN
                    WRITELN;
                    IF I > 39 THEN
                        WRITE('£    ',TRUNC(I/4),'    £     ---     £ ')
                    ELSE
                        WRITE('£     ',TRUNC(I/4),'    £     ---     £ ')
                END
            ELSE
                WRITE('    ---     £ ');
        END;
    END;
END;

(*-------------------------------------------------------------------------*)
```

```
EPROM Management System

PROCEDURE ENUM(X : INTEGER);
(*Puts on the EPROM number in the Map*)
BEGIN
    IF X < 63 THEN
    BEGIN
        IF X MOD 4 = Ø THEN
        BEGIN
            WRITELN;
            IF X > 39 THEN
                WRITE('£    ',TRUNC(X/4),'    £ ')
            ELSE
                WRITE('£     ',TRUNC(X/4),'    £ ')
        END;
    END;
END;

(*-------------------------------------------------------------------------*)
```

```
EPROM Management System

FUNCTION FIND_TOP : INTEGER;
(* Finds the highest block written to on the EPROM card*)
VAR
    I,J,X : INTEGER;

BEGIN
    X := Ø;
    FOR I := 1 TO 64 DO
    BEGIN
        FOR J := 1 TO 16 DO
        BEGIN
            IF (BUF[I].BLOC_LOC[J] > X) AND (BUF[I].BLOC_LOC[J] <> $E5) THEN
                X := BUF[I].BLOC_LOC[J];
        END;
    END;
    FIND_TOP := X;
END;

(*-------------------------------------------------------------------------*)
```

```
EPROM Management System

PROCEDURE DIREC( BASE : INTEGER);
(*Fills up the map until OFFSET and then writes 'DIRECTORY'*)
BEGIN
    IF BASE < 63 THEN
    BEGIN
        IF BASE > 4 THEN
        BEGIN
            FILL_BLOCK(Ø,BASE-2);
            ENUM(BASE-1);
        END;
        WRITE(' DIRECTORY  £ ');
    END;
END;

(*-------------------------------------------------------------------------*)
```

```
PROCEDURE DISPLAY_MAP(OFF,TOP_ENT : INTEGER);
(* Does the filling in of the EPROM map using the EPDIR file.*)
VAR     O,I,J,K,BLOCK_COUNT : INTEGER;
BEGIN
    O := OFF*4;
    BLOCK_COUNT := O+1;
    DIREC(BLOCK_COUNT);
    REPEAT
        I := 1;
        REPEAT
            J := 1;
            REPEAT
                J := J+1;
            UNTIL (J = 17) OR (BUF[I].BLOC_LOC[(J-1)] = (BLOCK_COUNT - O));
            I := I+1;
        UNTIL (I = (TOP_ENT+1)) OR
                  (BUF[(I-1)].BLOC_LOC[(J-1)] = (BLOCK_COUNT-O));
        IF (BUF[(I-1)].BLOC_LOC[(J-1)] = (BLOCK_COUNT-O)) THEN
        BEGIN
            FOR K := 1 TO 11 DO
            BEGIN
                WRITE(BUF[(I-1)].FILENAME[K]);
            END;
            WRITE(' £ ');
        END
        ELSE
            FILL_BLOCK(BLOCK_COUNT,BLOCK_COUNT);
        BLOCK_COUNT := BLOCK_COUNT + 1;
        ENUM(BLOCK_COUNT);
    UNTIL (BLOCK_COUNT = 64);
    WRITELN;
END;

(*-------------------------------------------------------------------------*)
```

```
PROCEDURE END_DISPLAY;
(*Puts the bottom line on the map*)
BEGIN
    WRITE('£----------£-------------£-------------£');
    WRITELN('-------------£-------------£');
END;
```

```
(*****************************************************************************)
(*              UTILITY PROCEDURES FOR FILE DELETE                           *)
(*****************************************************************************)

PROCEDURE KILL_IT(NAME,NAME1 : STRING; TOP_ENT : INTEGER);
(* Removes the directory entry.*)
VAR
    EXTENT,I,J    : INTEGER;
    DONE,MATCH : BOOLEAN;

BEGIN
    DONE := FALSE;
    I := 1;
    EXTENT := Ø;
    REPEAT
        REPEAT
            MATCH := TRUE;
            J := 1;
            REPEAT
                IF BUF[I].FILENAME[J] <> NAME[J] THEN
                    MATCH := FALSE;
                J := J+1;
            UNTIL (NOT MATCH) OR (J = 12);
            I := I+1;
        UNTIL (MATCH) OR (I = (TOP_ENT+1));
        IF MATCH THEN
        BEGIN
            FOR J := (I-1) TO TOP_ENT DO
                BUF[J] := BUF[(J+1)];
            DONE := TRUE;
            EXTENT := EXTENT + 1;
        END;
        I := I-1;
    UNTIL I = TOP_ENT;
```

```
    IF NOT DONE THEN
    BEGIN
        WRITELN;
        WRITELN('THERE IS NO ENTRY: ',NAME1);
        EXIT;
    END
    ELSE
    BEGIN
        WRITELN;
        WRITELN('RE-PROGRAM : EPDIR.COM');
        MAXENTRY := MAXENTRY - EXTENT;
    END;
END;
```

```
(*****************************************************************************)
(*              UTILITY PROCEDURES FOR FILE ADDITION                         *)
(*****************************************************************************)

FUNCTION FILE_SIZE(VAR X : FYLE) : INTEGER;
(*This function calculates the size of the file specified by X*)
VAR
    ENDED : BOOLEAN;
    RES,I : INTEGER;

BEGIN
    ENDED := FALSE;
    I := Ø;
    REPEAT
        BLOCKREAD(X,BUFR,RES,CPM_BLK,I);
        IF RES = Ø THEN
            I := I+1
        ELSE
            ENDED := TRUE;
    UNTIL ENDED;
    FILE_SIZE := I;
END;

(*---------------------------------------------------------------------------*)
```

```
FUNCTION CONV_SIZE(Y : INTEGER) : INTEGER;
(*Converts Y (a 128 byte block count) to a 2k block count*)
BEGIN
    IF (Y MOD 16) = Ø THEN
        CONV_SIZE := TRUNC(Y/16)
    ELSE
        CONV_SIZE := TRUNC(Y/16)+1;
END;

(*-------------------------------------------------------------------------*)
```

```
FUNCTION CHECK_SPACE(VAR DIR : FYL; NUM,OFF : INTEGER) : INTEGER;
(*Gives the position at which the new program must be loaded or 64 if there  *)
(*   is not enough space.*)
VAR
    TOTAL_ENT,J,K,L : INTEGER;
    NO  : BOOLEAN;

BEGIN
    TOTAL_ENT := READ_DIR(DIR);;
    SORT_ENTRIES(TOTAL_ENT);
    FOR J := 1 TO 64 DO
        PLOT[J] := FALSE;
    FOR J := 1 TO TOTAL_ENT DO
        FOR K := 1 TO 16 DO
            PLOT[BUF[J].BLOC_LOC[K]] := TRUE;
    FOR J := (65-(OFF*4)) TO 64 DO
        PLOT[J] := TRUE;
    WRITELN;
    WRITE('Enter eproms NOT available ');
    WRITE('(actual positions, -1<cr> if all available)> ');
```

```
    REPEAT
        READ(K);
        K := K-OFF;
        IF (K < 17) AND (K > -1) THEN
            FOR J := (K*4) TO ((K*4)+3) DO
                PLOT[J] := TRUE;
    UNTIL EOLN;
    WRITELN('THANKS');
    WRITELN;
    K := Ø;
    FOR J := 1 TO 64 DO
        IF NOT PLOT[J] THEN
            K := K+1;
    IF NUM < K THEN
    BEGIN
        L := Ø;
        REPEAT
            L := L + 1;
        UNTIL NOT PLOT[L];
        L := L - 1;
    END
    ELSE
        L := 64;
    CHECK_SPACE := L;
END;

(*-------------------------------------------------------------------------*)
```

```
FUNCTION SET_UP_ENTRY(TOP_ENT,BLK_1,S,S_2K : INTEGER) : INTEGER;
(*Sets up the entry in the CP/M format*)
VAR
    I,J,K,L,M,N,P,EXT : INTEGER;
    COPY_NF : STRING;

BEGIN
    I := TOP_ENT+1;
    COPY_NF := NEW_FILE;
    DELETE(COPY_NF,1,POS(':',COPY_NF));
    J := POS('.',COPY_NF);
    DELETE(COPY_NF,J,1);
    IF LENGTH(COPY_NF) < 11 THEN
        REPEAT
            INSERT(' ',COPY_NF,J);
        UNTIL LENGTH(COPY_NF) = 11;
    EXT := (TRUNC(S_2K/8)+1);
    N := EXT;
    P := S_2K;
    FOR J := 1 TO EXT DO
    BEGIN
        N := N-1;
        L := I+(J-1);
        BUF[L].DRIVE := $Ø;
        FOR K := 1 TO 11 DO
            BUF[L].FILENAME[K] := COPY_NF[K];
        BUF[L].SCRATCH1 := J-1;
        BUF[L].SCRATCH2 := Ø;
        BUF[L].SCRATCH3 := Ø;
```

```
        IF N = Ø THEN
            BUF[L].NUM_BLKS := S
        ELSE
        BEGIN
            BUF[L].NUM_BLKS := $8Ø;
            S := S - $81;
        END;
        M := Ø;
        REPEAT
            M := M+1;
        UNTIL (M > 64) OR (NOT PLOT[M]);
        IF N = Ø THEN
        BEGIN
            FOR K := 1 TO P DO
            BEGIN
                BUF[L].BLOC_LOC[2*K-1] := M;
                BUF[L].BLOC_LOC[2*K] := $ØØ;
                PLOT[M] := TRUE;
                REPEAT
                    M := M+1;
                UNTIL (NOT PLOT[M]) OR (M > 64);
            END;
            IF BUF[L].BLOC_LOC[16] = $E5 THEN
            FOR K := (P+1) TO 8 DO
            BEGIN
                BUF[L].BLOC_LOC[2*K-1] := $ØØ;
                BUF[L].BLOC_LOC[2*K] := $ØØ;
            END;
        END
```

```
        ELSE
        BEGIN
            FOR K := 1 TO 8 DO
            BEGIN
                BUF[L].BLOC_LOC[2*K-1] := M;
                BUF[L].BLOC_LOC[2*K] := $ØØ;
                PLOT[M] := TRUE;
                REPEAT
                    M := M+1;
                UNTIL (NOT PLOT[M]) OR (M > 64);
            END;
        END;
        P := P-8;
    END;
    SET_UP_ENTRY := EXT;
END;

(*-------------------------------------------------------------------------*)
```

```
PROCEDURE UPDATE_DIR(TOP_ENT,X : INTEGER);
(*Updates the EPDIR.COM program.*)
VAR I : INTEGER;

BEGIN
    FOR I := 1 TO X DO
    BEGIN
        INFILE := BUF[(TOP_ENT+I)];
        SEEKWRITE(INFILE,(TOP_ENT+I-1));
    END;
    IF IORESULT <> Ø THEN
    BEGIN
        WRITELN('ERROR IN EXTENDING EPDIR.COM');
        EXIT;
    END;
END;

(*-------------------------------------------------------------------------*)
```

```
EPROM Management System

PROCEDURE WHAT_FILE(BLK_I : INTEGER);
(*Opens the EPn.COM on the basis of I.*)
BEGIN
    CASE TRUNC(BLK_I/4) OF
        Ø : EP_FILE := 'EPDIR.COM';
        1 : EP_FILE := 'EP1.COM';
        2 : EP_FILE := 'EP2.COM';
        3 : EP_FILE := 'EP3.COM';
        4 : EP_FILE := 'EP4.COM';
        5 : EP_FILE := 'EP5.COM';
        6 : EP_FILE := 'EP6.COM';
        7 : EP_FILE := 'EP7.COM';
        8 : EP_FILE := 'EP8.COM';
        9 : EP_FILE := 'EP9.COM';
        1Ø : EP_FILE := 'EP1Ø.COM';
        11 : EP_FILE := 'EP11.COM';
        12 : EP_FILE := 'EP12.COM';
        13 : EP_FILE := 'EP13.COM';
        14 : EP_FILE := 'EP14.COM';
        15 : EP_FILE := 'EP15.COM';
    END;
END;

(*-------------------------------------------------------------------------*)
```

```
PROCEDURE NEW_EPROM(VAR INF : FYLE; TOP_ENT,EX,S,S_2K : INTEGER);
(*Creates the EPn.COM programs and saves them on disk*)
VAR
    X,I,J,K,BLK,RES,C_BLK : INTEGER;
    TRANS_BUF : ARRAY[1..CPM_BLK] OF BYTE;
    OEFILE,EFILE : STRING;
    S_128,M : BYTE;

BEGIN
    WRITELN;
    WRITELN('BURN THE FOLLOWING PROGRAMS INTO EPROMS:');
    WRITELN('EPDIR.COM');
    RESET(INF);
    X := 0;
    FOR J := 1 TO EX DO
    BEGIN
        I := 1;
        BLK := BUF[TOP_ENT+J].BLOC_LOC[I];
        S_128 := BUF[TOP_ENT+J].NUM_BLKS;
        M := 0;
        WHAT_FILE(BLK);
        OEFILE := EP_FILE;
        ASSIGN(E,EP_FILE);
        RESET(E);
        REPEAT
            BLK := BUF[TOP_ENT+J].BLOC_LOC[I];
            WHAT_FILE(BLK);
            EFILE := EP_FILE;
```

```
    IF EFILE <> OEFILE THEN
    BEGIN
        CLOSE(E,RES);
        IF RES <> Ø THEN
        BEGIN
            WRITELN('ERROR IN CLOSING ',OEFILE);
            EXIT;
        END;
        WRITELN(OEFILE); (* EPROM to be programmed.*)
        ASSIGN(E,EFILE);
        RESET(E);
        OEFILE := EFILE;
    END;
    C_BLK := (BLK MOD 4)*16; (* C_BLK has the CP/M block number.*)
    REPEAT
        BLOCKREAD(INF,TRANS_BUF,RES,CPM_BLK,X);
        IF RES <> Ø THEN
            M := S_128; (* To show end of data.*)
        IF M < S_128 THEN
        BEGIN
            BLOCKWRITE(E,TRANS_BUF,RES,CPM_BLK,C_BLK);
            IF RES <> Ø THEN
            BEGIN
                WRITELN('DISK FULL');
                EXIT;
            END;
        END;
        X := X+1;
        M := M+1;
        C_BLK := C_BLK + 1;
    UNTIL (C_BLK MOD 16 = Ø) OR (M > S_128);
```

```
            I := I+2;
        UNTIL (BUF[TOP_ENT+J].BLOC_LOC[I] = Ø) OR (I > 16) OR (M > S_128);
        CLOSE(E,RES);
        WRITELN(EFILE);
    END;
END;

(*-------------------------------------------------------------------------*)
```

```
FUNCTION THERE(ADD_FILE : STRING; TOP_ENT : INTEGER) : BOOLEAN;
(* Checks to see if the file specified by ADD_FILE is in the directory.*)
VAR
    I,J : INTEGER;
    Y : BOOLEAN;
    A_FILE : STRING;
    REP : CHAR;

BEGIN
    A_FILE := ADD_FILE;
    DELETE(A_FILE,1,POS(':',A_FILE));
    J := POS('.',A_FILE);
    DELETE(A_FILE,J,1);
    IF LENGTH(A_FILE) < 11 THEN
        REPEAT
            INSERT(' ',A_FILE,J);
        UNTIL LENGTH(A_FILE) = 11;
    I := 1;
    REPEAT
        Y := TRUE;
        J := 1;
        REPEAT
            IF BUF[I].FILENAME[J] <> A_FILE[J] THEN
                Y := FALSE;
            J := J+1;
        UNTIL (NOT Y) OR (J = 12);
        I := I+1;
    UNTIL (Y) OR (I = (TOP_ENT+1));
```

```
    IF Y THEN
    BEGIN
        WRITELN('There is another ',ADD_FILE,' in memory already,');
        WRITE('D)elete it or R)eturn to main menu [D/R] ');
        READLN(REP);
        IF REP = 'D' THEN
        BEGIN
            KILL_IT(A_FILE,A_FILE,TOP_ENT);
            J := TOP_ENT;
            TOP_ENT := Ø;
            UPDATE_DIR(TOP_ENT,J);
            THERE := TRUE;
        END
        ELSE
            THERE := FALSE;
    END
    ELSE
        THERE := TRUE;
END;
```

EPROM Management System

```
(*****************************************************************************)
(*      MAJOR PROCEDURES                                                     *)
(*****************************************************************************)

PROCEDURE EPROM_MAP(OFFS : INTEGER);
(*This is the procedure that outputs the map of data in the EPROM card*)
VAR
    RESP      : CHAR;
    TOTAL_ENT : INTEGER;

BEGIN
    OPEN(INFILE,'EPDIR.COM',RESULT);
    IF RESULT = 255 THEN
    BEGIN
        WRITELN('NO EPDIR PRESENT ON THIS DISK');
        @HLT;
    END
    ELSE
    BEGIN
        TOTAL_ENT := READ_DIR(INFILE);
        SORT_ENTRIES(TOTAL_ENT);
        SET_UP_CON;
        DISPLAY_MAP(OFFS,TOTAL_ENT);
        END_DISPLAY;
        WRITE('Hit <CR> to continue');
        READLN(RESP);
    END;
END;

(*-----------------------------------------------------------------------*)
```

```
PROCEDURE FILE_ADD(TOTAL_ENT : INTEGER);
(*This is the main controlling procedure for file addition.*)
VAR
    CONT : BOOLEAN;
    SIZ_128,SIZ_2K,SPCE,EXT  : INTEGER;

BEGIN
    WRITELN;
    WRITELN;
    WRITELN('                          FILE ADDITION UTILITY');
    WRITELN('                          ====================');
    WRITELN;
    WRITE('NAME OF NEW FILE TO ADD > ');
    READLN(NEW_FILE);
    CONT := THERE(NEW_FILE,TOTAL_ENT);
    IF CONT THEN
    BEGIN
        OPEN(N,NEW_FILE,RESULT);
        IF RESULT <> Ø THEN
        BEGIN
            WRITELN('THERE IS NO ',NEW_FILE);
            EXIT;
        END;
        ASSIGN(INFILE,'EPDIR.COM');
        RESET(INFILE);
        IF IORESULT <> Ø THEN
        BEGIN
            WRITELN('THERE IS NO EPDIR.COM ON THIS DISK');
            @HLT;
        END;
```

```
        SIZ_128 := FILE_SIZE(N); (*Return the number of 128 byte blocks in N.*)
        SIZ_2K := CONV_SIZE(SIZ_128); (* Return the number of 2k blocks in N.*)
        SPCE := CHECK_SPACE(INFILE,SIZ_2K,OFFSET); (* Ø-63 is block number,
                                                    64 = error.*)
        IF SPCE = 64 THEN
        BEGIN
            WRITELN('THERE IS NOT ENOUGH SPACE FOR THAT FILE IN MEMORY');
            EXIT;
        END;
        TOTAL_ENT := MAXENTRY;
        EXT := SET_UP_ENTRY(TOTAL_ENT,SPCE,SIZ_128,SIZ_2K); (* Sets up
                            directory entry, returns the number of extents.*)
        MAXENTRY := MAXENTRY + EXT;
        UPDATE_DIR(TOTAL_ENT,EXT); (* Update the directory file.*)
        CLOSE(INFILE,RESULT);
        IF RESULT <> Ø THEN
        BEGIN
            WRITELN('CANNOT CLOSE EPDIR.COM');
            EXIT;
        END;
        NEW_EPROM(N,TOTAL_ENT,EXT,SIZ_128,SIZ_2K); (* Creates new EPn.com.*)
        CLOSE(N,RESULT);
        IF RESULT <> Ø THEN
        BEGIN
            WRITELN('CANNOT CLOSE ',NEW_FILE);
            EXIT;
        END;
    END;
END;

(*-------------------------------------------------------------------------*)
```

```
EPROM Management System

PROCEDURE FILE_DELETE;
(* Removes the entry from the File Directory.*)
VAR
    TOTAL_ENT,J,TEM : INTEGER;
    DUP : STRING;

BEGIN
    WRITELN;
    WRITELN;
    WRITELN('                          FILE DELETION UTILITY');
    WRITELN('                          ====================');
    WRITELN;
    WRITE('DELETE WHICH FILE? ');
    READLN(ANS);
    DUP := ANS;
    DELETE(ANS,1,POS(':',ANS));
    J := POS('.',ANS);
    IF J = Ø THEN
        J := LENGTH(ANS)
    ELSE
        DELETE(ANS,J,1);
    IF LENGTH(ANS) < 11 THEN
        REPEAT
            INSERT(' ',ANS,J);
        UNTIL LENGTH(ANS) = 11;
    ASSIGN(INFILE,'EPDIR.COM');
    RESET(INFILE);
```

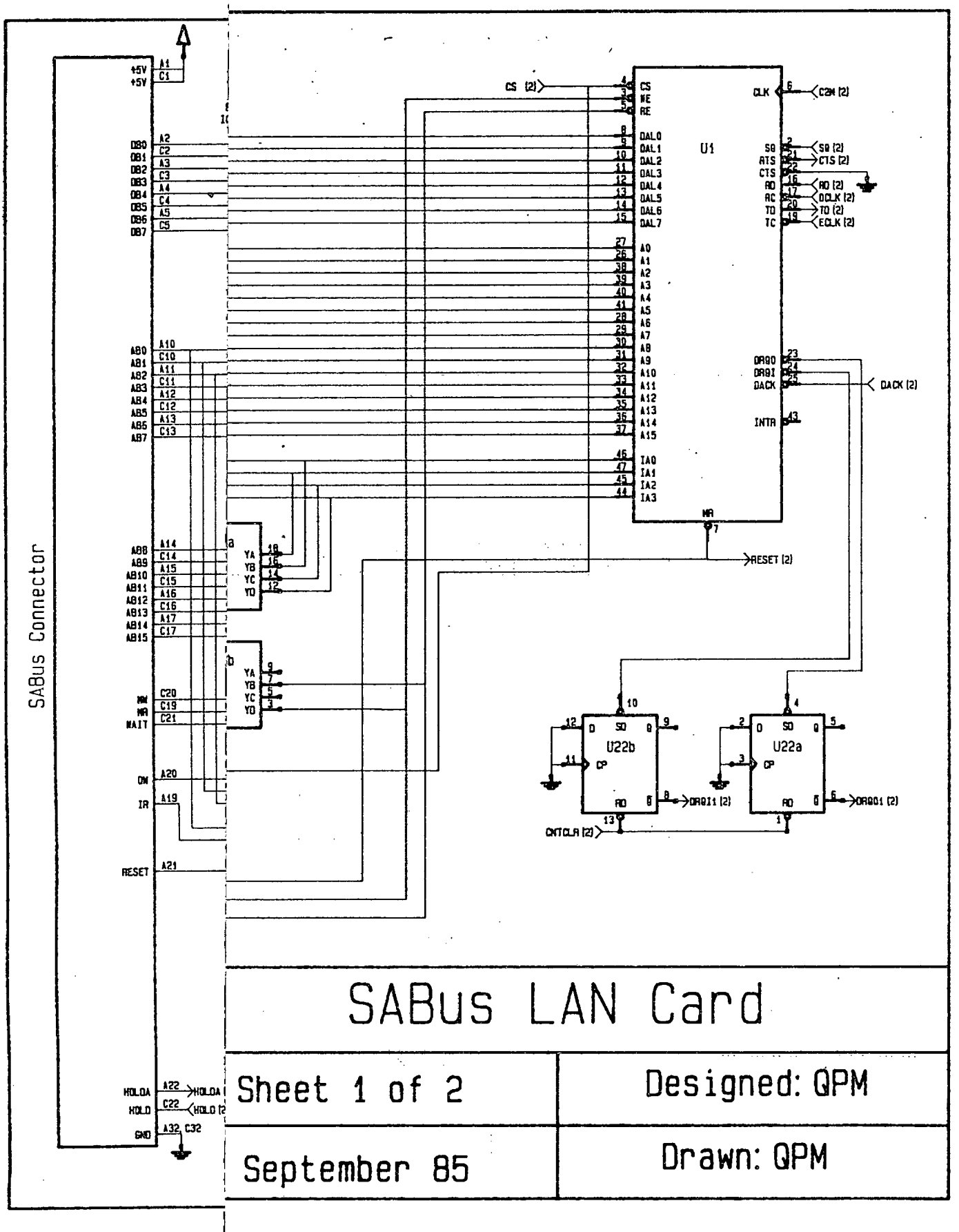
```
    IF IORESULT <> Ø THEN
    BEGIN
        WRITELN('THERE IS NO EPDIR.COM ON THIS DISK');
        @HLT;
    END;
    TOTAL_ENT := READ_DIR(INFILE);
    KILL_IT(ANS,DUP,TOTAL_ENT);
    TEM := TOTAL_ENT;
    TOTAL_ENT := Ø;
    UPDATE_DIR(TOTAL_ENT,TEM);
    WRITELN;
END;
```

```
(*************************************************************************)
(*      MAIN LOOP                                                        *)
(*************************************************************************)

BEGIN
    OPEN(INFILE,'EPDIR.COM',RESULT);
    IF RESULT = 255 THEN
    BEGIN
        WRITELN('NO EPDIR.COM ON THIS DISK');
        @HLT;
    END;
    MAXENTRY := READ_DIR(INFILE);
    OFFSET := 0;
    REPEAT
        WRITELN;
        WRITELN;
        WRITELN('                          EPROM MANAGEMENT SYSTEM');
        WRITELN('                          ======================');
        WRITELN;
        WRITELN(' Main menu: ');
        WRITELN;
        WRITELN('Choose one of the following:');
        WRITELN('A)dd file');
        WRITELN('D)elete file');
        WRITELN('M)ap of files');
        WRITELN('R)eturn to CP/M');
        WRITELN;
        READLN(RESPONSE);
```

```
        CASE RESPONSE OF
            'A','a' : FILE_ADD(MAXENTRY);
            'D','d' : FILE_DELETE;
            'R','r' : @HLT;
            'M','m' : EPROM_MAP(OFFSET)
        ELSE
            WRITELN('Enter ONLY A, D, M or R please')
        END;
    UNTIL FALSE;
END.
```

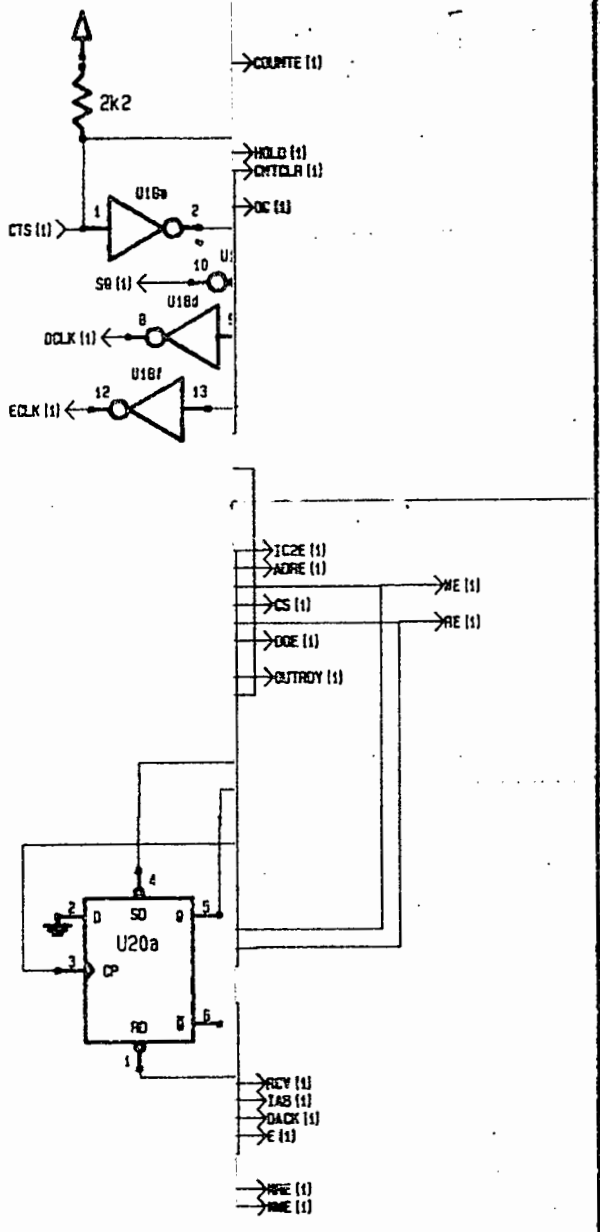
APPENDIX D

SABus LAN Card Circuit Diagrams



SABus LAN Card

Sheet 1 of 2	Designed: QPM
September 85	Drawn: QPM



Power Table

IC#	Type	Power	GND
1	WD2840	48 (5V) 42 (12V)	18
2	74LS245	20	10
3	74LS373	20	10
4	74LS373	20	10
5	74LS367	16	8
6	74LS85	16	8
7	74LS367	16	8
8	74LS374	20	10
9	74LS244	20	10
10	PAL20L10	24	12
11	PAL20L10	24	12
12	PAL20L10	24	12
13	74LS164	14	7
14	74LS164	14	7
15	74LS164	14	7
16	555	8	1
17	HD6409	20	10
18	74LS04	14	7
19	74LS93	5	10
20	74LS74	14	7
21	75176	8	5
22	74LS74	14	7

SABus LAN Card

Sheet 2 of 2	Designed: QPM
September 85	Drawn: QPM

The PAL equations for the SABus LAN card are given below. As before a / means negative logic.

PAL 1

```
/OC = /WDRDY * T1 * /T13

/CNTCLR = T23 * /C16M  +  /RESET

/HOLD = /DRQI * /T13  +  /DRQO * /T13

/P = C16M * /C1SEC * HOSTCS * WDRDY  +
       C16M * /C1SEC * T1 * WDRDY

/N = C16M * INRDY * /DRQO  +  HOLD * HOLDA  +
       C16M * /INRDY * /DRQO * T1Ø

/N = C16M * INRDY * /DRQI  +  HOLD * HOLDA  +
       C16M * /INRDY * /DRQI * T13

/H = /HOLD * HOLDA

/COUNTE = M  +  N  +  P
```

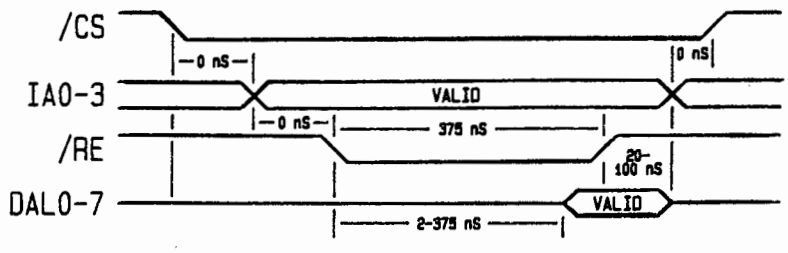
PAL 2

```
/WDRDY = /WRCY * /RDCY

/NARCY = /C1SEC * WDRDY * /HOSTCS  +
           TØ * /T23 * /WDRDY * /C1SEC

/OUTRDY = HOSTCS * WDRDY  +  HOSTCS * NARCY

/DOE = NARCY * T19 * /T23

/RE = NARCY * T4 * /T1Ø
```

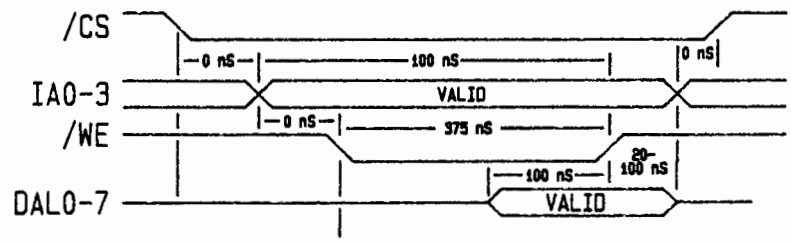
```
/WE = NARCY * T15 * /T22

/CS = NARCY * TØ * /T1Ø   +  NARCY * T14 * T23  +
        /NARCY * HOSTCS

/ADRE = NARCY * TØ * /T23

/IC2E = /NARCY
```

PAL 3

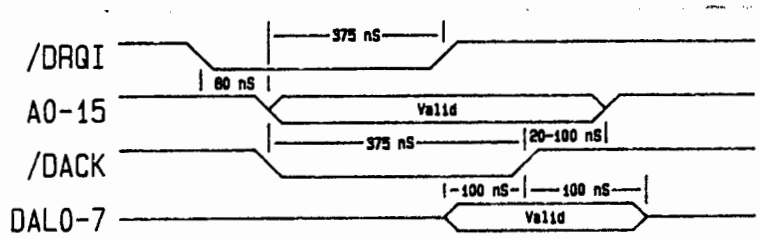
```
/RDCY = /DRQI * T1 * /IC2E

/WRCY = /DRQO * T1 * /IC2E

/MWE = WRCY * T6 * /T1Ø

/MRE = RDCY * T6 * T13

/DACK = T1 * /T12 * WRCY  +
          T1 * /T11 * RDCY

/E = /DACK * /OC

/IAB = /RE * /WE  +  RE * WE

/RCV = WRCY * /OC  +  HOSTCS * /RE
```



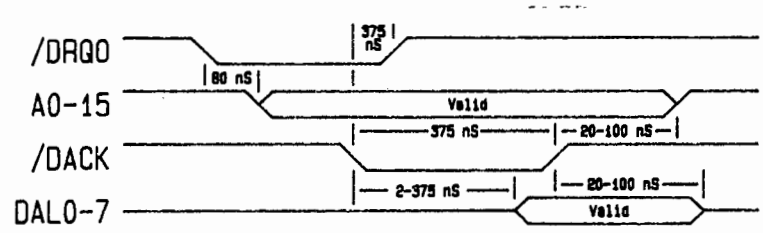
WD2840 Control/Status Register Read Timing



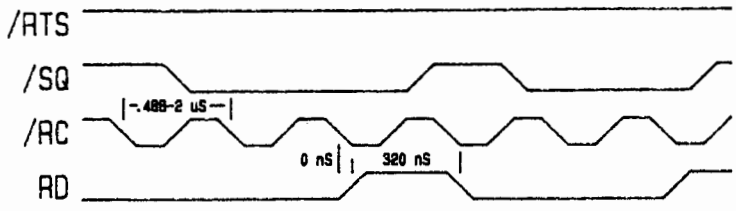
WD2840 Control/Status Register Write Timing



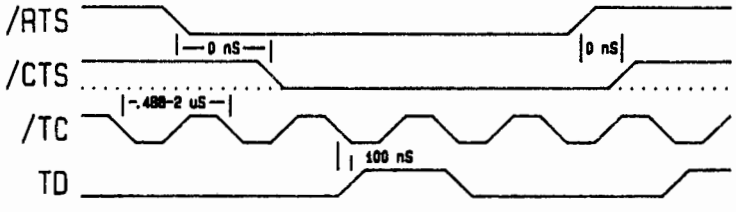
WD2840 DMA Input Timing



WD2840 DMA Output Timing



WD2840 Network Read Timing



WD2840 Network Write Timing

APPENDIX E

The TAC User Routines Program

```
LOC  OBJ        LINE        SOURCE STATEMENT

                    1 $PAGELENGTH(35)
                    2 ;$OBJECTFILE(TACUR.H85)
                    3 ;*************************************************************
                    4 ;
                    5 ;                     USER ROUTINES FOR TAC
                    6 ;
                    7 ;*************************************************************
                    8 ;
                    9 ;       PROGRAM:        TACUR.A85
                   10 ;       TARGET SYSTEM:  UCT SABus Micro-computer
                   11 ;       AUTHOR:         Q.P. Mc Grath
                   12 ;       DATE:           19 September 1985
                   13 ;
                   14 ; The Token Access Controller (TAC), WD2840, situated at address
                   15 ; 70H to 7FH, is used to control the access to the network.
                   16 ; This program enables the user to access the network using  the
                   17 ; three routines, NTWRKINIT, which does the network initialisation,
                   18 ; SNDMSG, which will transmit a frame pointed to by the DE registers,
                   19 ; and RECMSG which receives a frame from the network and writes it
                   20 ; to a buffer in memory returning the address in the DE registers.
                   21 ;
                   22 ; The receive and transmit format is the same:
                   23 ;       Length (Least significant byte)
                   24 ;       Length (Most significant byte)
                   25 ;       Destination Address (Tx frame) or Source Address (Rx frame)
                   26 ;       Data bytes, length bytes long
                   27 ;
                   28 ; A fourth procedure, STAT, is supplied which returns the status
                   29 ; of the network.  Here, DE point at a buffer which contains a
                   30 ; copy of the 16 registers on th WD2840 as well as a copy of the
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                  31 ; 11 event registers.  Lastly, the software enables the higher
                  32 ; layers to instruct it off the network by using the STNDWN
                  33 ; entry point.
                  34 ;
                  35 ; This software is based on the IEEE 802 standard specifications
                  36 ; using a Class I LLC layer which only implements an unacknowledged
                  37 ; connectionless transmission or as it is more commonly known a
                  38 ; datagram service.  The implementation also uses a simple PHY and MAC
                  39 ; check, which involves verifying byte correctness and, in the case of an
                  40 ; error the program will inform the user.  It will also inform the
                  41 ; user of the reason for the error in this case.
                  42 ;
                  43 ; The addressing is done on a one to one basis throughout the
                  44 ; PHY, MAC  and LLC Layers.  This limits the addressing to only
                  45 ; 63 nodes as defined by the LLC SAPs.
                  46 ;
                  47 ; Data is transmitted using the IEEE 802 LLC frame format, although
                  48 ; the WD2840 does not allow the same PHY frame format.  Thus the
                  49 ; entry to this program looks like a LLC/Network layer interface
                  50 ; as defined by IEEE.
                  51 ;
                  52 ; Responses to both the TEST frame and the XID, exchange
                  53 ; Identification, control frames are supported by this program.
                  54 ; To send a control frame the value of the B register is set to
                  55 ; 0FFH and then the SNDMSG procedure does not add the UI,
                  56 ; Unnumbered Information, control word but simply transmits the
                  57 ; data found in the buffer pointed to by the DE register pair.
                  58 ; The program responds to the TEST and XID frames at the earliest
                  59 ; possible chance.
                  60 ;
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                    61 ;=================================================================
                    62 $EJECT
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                    63 ;=============================================================
                    64 ;
                    65 ;                              Equates
                    66 ;
                    67 ;=============================================================
                    68 ;-------------------------------------------------------------
                    69 ;
                    70 ;                         True or False Equates
                    71 ;
                    72 ;-------------------------------------------------------------
                    73
0000                74 FALSE    EQU     0
FFFF                75 TRUE     EQU     NOT FALSE
                    76
                    77 ;-------------------------------------------------------------
                    78 ;
                    79 ;                 General Equates
                    80 ;
                    81 ;-------------------------------------------------------------
                    82
000A                83 LF       EQU     10      ; Line Feed
000D                84 CR       EQU     13      ; Carriage Return
001A                85 CTRLZ    EQU     26      ; Control Z
                    86
                    87 ;-------------------------------------------------------------
                    88 ;
                    89 ;                 Monitor Call equates
                    90 ;
                    91 ;-------------------------------------------------------------
                    92
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

F03D                93 OUTMSG  EQU     0F03DH  ; Output a message routine
F00A                94 CITEST  EQU     0F00AH  ; Test for console input
F013                95 CI      EQU     0F013H  ; Console input routine
F043                96 DCMP    EQU     0F043H  ; 16 bit compare
                    97
                    98 ;---------------------------------------------------------------
                    99 ;
                   100 ;                Controller Equates
                   101 ;
                   102 ;---------------------------------------------------------------
                   103
0070               104 NETCTRL EQU      070H          ; Network Controller Base address
                   105
0070               106 CR0     EQU      00H+NETCTRL   ; Control Register 0
0071               107 CR1     EQU      01H+NETCTRL   ; Control Register 1
0072               108 SR0     EQU      02H+NETCTRL   ; Status Register 0
0074               109 SR1     EQU      04H+NETCTRL   ; Status Register 1
0075               110 SR2     EQU      05H+NETCTRL   ; Status Register 2
0073               111 IR0     EQU      03H+NETCTRL   ; Interrupt Type Register
0076               112 CTR0    EQU      06H+NETCTRL   ; Counter Register
0077               113 NA      EQU      07H+NETCTRL   ; Next Address Register
0078               114 TA      EQU      08H+NETCTRL   ; ACK return timer
0079               115 TD      EQU      09H+NETCTRL   ; Network dead timer
007A               116 CBPH    EQU      0AH+NETCTRL   ; Control Block Pointer (MSB)
007B               117 CBPL    EQU      0BH+NETCTRL   ; Control Block Pointer (LSB)
007C               118 NAR     EQU      0CH+NETCTRL   ; Next Address Requester
007D               119 AHOLT   EQU      0DH+NETCTRL   ; Access Hold-Off Limit
007E               120 TXLT    EQU      0EH+NETCTRL   ; Transmission Limit
007F               121 MA      EQU      0FH+NETCTRL   ; My Address
                   122
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

0010              123 RESPONSETIME  EQU  010H        ; Allows 500uS for ACK to return
0080              124 TIMEOUT       EQU  80H         ; Allows time for transmission
0000              125 CYCLESKIP     EQU  0           ; Number of cycles skipped on entry
0001              126 MAXFRAMES     EQU  1           ; Max allowable frames/token
0002              127 THISNODE      EQU  2           ; The value written to My Address
0004              128 BUFLEN        EQU  4           ; Sets up the buffer length to 320 bytes
0140              129 BYTECOUNT     EQU  (BUFLEN+1)*64 ; Number of bytes in a buffer
                  130
00C0              131 FCBWAIT EQU     11000000B      ; Wait for acknowledge and last frame
0000              132 FCBNOWT EQU     00000000B      ; Do not wait for acknowledge
                  133
                  134 ;-----------------------------------------------------------------
                  135 ;
                  136 ;                Frame Equates
                  137 ;
                  138 ;-----------------------------------------------------------------
                  139
0003              140 UIFR    EQU     00000011B      ; The Unnumbered frame control word
0003              141 UI      EQU     UIFR           ;
00AF              142 XID     EQU     10101111B      ; Exchange ID control byte
00E3              143 TEST    EQU     11100011B      ; TEST control word
0010              144 POLLB   EQU     00010000B      ; The poll bit
00EF              145 NOPOLL  EQU     11101111B      ; Inverse poll
                  146
                  147 ;-----------------------------------------------------------------
                  148 ;
                  149 ;                      Hardware Equates
                  150 ;
                  151 ;-----------------------------------------------------------------
                  152
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

00B0              153 CONTWD        EQU  10110000B ; Control word for the USART
000B              154 CONT8253      EQU  0BH       ; Address of USART control and status
000A              155 C28253        EQU  0AH       ; Address of the data register
                  156 $EJECT
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     169 ;-----------------------------------------------------------
                     170 ;                      Network Initialization
                     171 ;-----------------------------------------------------------
                     172 ; FUNCTION: NTWRKINIT
                     173 ; INPUTS : None
                     174 ; OUTPUTS: CY : set means error
                     175 ;             : clear means no error
                     176 ; DESTROYS: A, B, C, D, E, H, L, Flags
                     177 ; DESCRIPTION:
                     178 ;            NTWRKINIT sets up the registers in the WD2840, it
                     179 ;            will then set up the control block.  Once the network
                     180 ;            has been checked for duplicate nodes it will attempt
                     181 ;            enter the transmission ring.  It will return carry
                     182 ;            set if there is an error or clear if no error.
                     183 ;
                     184 ;-----------------------------------------------------------
                     185 NTWRKINIT:
                     186
                     187 ; Set up the Network parameters  (* Station in the DOWN STATE *)
                     188
 000F  3E02          189         MVI     A,THISNODE      ; Only this value must be changed on each
                     190                                 ;      TAC
 0011  32420A    C   191         STA     MYID            ; Save the ID
 0014  212F05    C   192         LXI     H,CBLOCK        ; Set up the buffer pointers
 0017  113F05    C   193         LXI     D,RXBUF1        ;    in the Control Block
 001A  72            194         MOV     M,D             ;
 001B  23            195         INX     H               ;
 001C  73            196         MOV     M,E             ;    (MSB then LSB)
 001D  23            197         INX     H               ;
 001E  11BF07    C   198         LXI     D,TXBUF1        ; For both receive and transmit buffers
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

0021  72             199          MOV     M,D              ;
0022  23             200          INX     H                ;
0023  73             201          MOV     M,E              ;
0024  23             202          INX     H                ;
0025  3604           203          MVI     M,BUFLEN         ; Set up the buffer size
                     204
                     205 ; Then set up the Register values
                     206
0027  3E10           207          MVI      A,RESPONSETIME  ; - The length waited for Ack
0029  D378           208          OUT      TA              ; Write to the rquired addr
002B  0680           209          MVI      B,TIMEOUT       ; - Network Dead Time
002D  3A420A    C    210          LDA      MYID            ;     calculated from MA value
0030  B7             211          ORA      A               ;
0031  1F             212          RAR                      ; Devide by two
0032  80             213          ADD      B               ;
0033  D379           214          OUT      TD              ;
0035  212F05    C    215          LXI      H,CBLOCK        ; Fetch the Control Block addr
0038  7D             216          MOV      A,L             ; - Output the lower address
0039  D37B           217          OUT      CBPL            ;
003B  7C             218          MOV      A,H             ; - Then the higher address
003C  D37A           219          OUT      CBPH            ;
003E  3A420A    C    220          LDA      MYID            ; - Set up the Next Address
0041  3C             221          INR      A               ;
0042  D37C           222          OUT      NAR             ;
0044  3E00           223          MVI      A,CYCLESKIP     ; - Miss the required cycles
0046  D37D           224          OUT      AHOLT           ;
0048  3E01           225          MVI      A,MAXFRAMES     ; - Store max frames/token
004A  D37E           226          OUT      TXLT            ;
004C  3A420A    C    227          LDA      MYID            ; - My address
004F  D37F           228          OUT      MA              ;
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     229
                     230 ; Dummy Start, checking for Network dead
                     231
0051  3E00           232         MVI     A,00            ; Set the two control regs to zero
0053  D371           233         OUT     CR1             ;
0055  D370           234         OUT     CR0             ;
                     235 STCHG:                          ;
0057  DB75           236         IN      SR2             ; Wait for the State Change
0059  E602           237         ANI     02H             ;     - indicates ready
005B  E3             238         XTHL                    ; Create a delay for the WD
005C  E3             239         XTHL                    ;   to update its registers
005D  C25700    C    240         JNZ     STCHG           ;
                     241                                 ;
0060  0EFF           242         MVI     C,0FFH          ; Counter for Software loop
                     243 LOOPD:                          ;
0062  DB73           244         IN      IR0             ; Is the network dead?
0064  E601           245         ANI     01H             ; Set if time TD passes
0066  C2A000    C    246         JNZ     DEAD            ;
0069  E3             247         XTHL                    ; Delay for WD reg update
006A  E3             248         XTHL                    ;
006B  0D             249         DCR     C               ;
006C  C26200    C    250         JNZ     LOOPD           ; If C = 0 then the Network is not dead
                     251                                 ;
                     252 NONDEAD:                        ;
006F  3E01           253         MVI     A,01H           ; - Take NAR
0071  D371           254         OUT     CR1             ;
0073  3E10           255         MVI     A,10H           ; - Enable Token Interrupts
0075  D370           256         OUT     CR0             ;
                     257
                     258 ; Watch the ITOK for addr dup    (* Station in DUPLICATE ADDRESS CHECK STATE *)
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     259
0077   015046        260         LXI     B,18000         ; This will keep looking for 0.5 sec
007A   3E00          261         MVI     A,0             ; Reset counter
007C   32500A    C   262         STA     ADDUP           ;
                     263 DIAG1:                          ;
007F   DB73          264         IN      73H             ; Read interrupt register
0081   E604          265         ANI     04H             ; ITOK bit set?
0083   C29100    C   266         JNZ     FDIAG           ;
0086   E3            267         XTHL                    ; Let WD update registers
0087   E3            268         XTHL                    ;
0088   0B            269         DCX     B               ;
0089   78            270         MOV     A,B             ; Time up?
008A   B1            271         ORA     C               ;
008B   C27F00    C   272         JNZ     DIAG1           ;
008E   C3A000    C   273         JMP     DEAD            ;
                     274 FDIAG:                          ;
0091   3A500A    C   275         LDA     ADDUP           ; Can only be a duplicate frame
0094   3C            276         INR     A               ;   if it happens more than once
0095   FE02          277         CPI     2               ;   as it could be a scan
0097   CADD00    C   278         JZ      FAILSDIAG       ;
009A   32500A    C   279         STA     ADDUP           ;
009D   C37F00    C   280         JMP     DIAG1           ;
                     281
                     282 ; Get into the logical loop
                     283
                     284 DEAD:                           ;
00A0   DB71          285         IN      CR1             ; Get into the logical loop
00A2   F628          286         ORI     28H             ;
00A4   D371          287         OUT     CR1             ;
00A6   3E50          288         MVI     A,50H           ; RXEN set
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

00A8  D370           289          OUT     CR0              ;
                     290 READY1:                           ;
00AA  DB75           291          IN      SR2              ; STATE clear confirms change
00AC  E603           292          ANI     03H              ; IRING set means token passed
00AE  FE01           293          CPI     1                ;
00B0  F5             294          PUSH    PSW              ;
00B1  CD0101    C    295          CALL    DLAY             ;
00B4  F1             296          POP     PSW              ;
00B5  C2AA00    C    297          JNZ     READY1           ;
                     298
                     299 ; Enable the transmission of the frames
                     300
00B8  3E42           301          MVI     A,042H           ; Enable transmission, disable
00BA  D370           302          OUT     CR0              ;   ITOK interrupts
                     303
                     304 ; Set up buffer pointers
                     305
00BC  3E02           306          MVI     A,2              ; Do the required initialisation
00BE  32400A    C    307          STA     ?TXBUF           ;    of memory
00C1  323F0A    C    308          STA     ?RXBUF           ; Set up Buffer-to-use pointer
00C4  3E00           309          MVI     A,0              ;
00C6  324105    C    310          STA     RXBUF1+2         ; Reset receive buffer FSBs
00C9  328106    C    311          STA     RXBUF2+2         ;
00CC  3E80           312          MVI     A,80H            ; Set transmit buffer FSBs to DONE
00CE  32C107    C    313          STA     TXBUF1+2         ;
00D1  320109    C    314          STA     TXBUF2+2         ;
                     315
                     316 ; Finish                         (* Station in UP STATE *)
                     317
00D4  21CB03    C    318          LXI     H,READY          ; Inform the user
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

 00D7  CD3DF0        319          CALL    OUTMSG          ;
                     320                                  ;
 00DA  37            321          STC                     ; As flag
 00DB  3F            322          CMC                     ;
 00DC  C9            323          RET                     ;
                     324
                     325 ; Exit here if error            (* Station in DOWN STATE *)
                     326
                     327 FAILSDIAG:                       ;
 00DD  21B103    C   328          LXI     H,FAILMSG       ; Tell user
 00E0  CD3DF0        329          CALL    OUTMSG          ;
                     330                                  ; And let it continue and bring the
                     331                                  ;     station down
                     332
                     333 $EJECT
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     334 ;-----------------------------------------------------------
                     335 ;                    Brings the station off-line
                     336 ;-----------------------------------------------------------
                     337 ; FUNCTION: STNDWN
                     338 ; INPUTS  : None
                     339 ; OUTPUTS : A = 0 the isolation confirmation is pending
                     340 ;           A = 2 the isolation is complete
                     341 ;           Carry Bit set
                     342 ; DESTROYS: A, B, Flags
                     343 ; DESCRIPTION:
                     344 ;             Sets the WD2840 into off-line state and waits for
                     345 ;             200 mS for confirmation after which it will set
                     346 ;             the accumulator to indicate confirmation pending
                     347 ;             and return.  If the confirmation occurs in the
                     348 ;             200 mS provided then the accumulator returns with
                     349 ;             a value of 2.  The carry bit is set to enable
                     350 ;             compatability with the CP/NET status returns.
                     351 ;
                     352 ;-----------------------------------------------------------
                     353 STNDWN:                              ; Bringing station down
00E3  3E01           354         MVI     A,01H               ; Instruct the WD2840 to isolate
00E5  D370           355         OUT     CR0                 ;
00E7  0614           356         MVI     B,20                ;
                     357 WAITONSTAT:                         ;
00E9  CD0101    C    358         CALL    DLAY                ; Wait for WD to update registers
00EC  DB75           359         IN      SR2                 ;
00EE  E602           360         ANI     02H                 ; SR22 set confirms in ISOL state
00F0  C2F900    C    361         JNZ     DWNITIS             ;
00F3  05             362         DCR     B                   ;
00F4  C2E900    C    363         JNZ     WAITONSTAT          ;
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

00F7  3E00           364          MVI     A,0              ; Indicate pending state
                     365 DWNITIS:                          ;
00F9  21DD03    C    366          LXI     H,OUTITIS        ; Inform user
00FC  CD3DF0         367          CALL    OUTMSG           ;
00FF  37             368          STC                      ; Set the carry as a flag
0100  C9             369          RET                      ;
                     370
                     371 $EJECT
```

```
 LOC   OBJ        LINE      SOURCE STATEMENT

                   372 ;-----------------------------------------------------------
                   373 ;                      10ms Delay
                   374 ;-----------------------------------------------------------
                   375 ; FUNCTION: DLAY
                   376 ; INPUTS  : None
                   377 ; OUTPUTS : None
                   378 ; DESTROYS: None
                   379 ; DESCRIPTION:
                   380 ;              Delays the cpu for 10 mS.
                   381 ;
                   382 ;-----------------------------------------------------------
                   383 DLAY:                              ; Delays for 10 mS
0101  C5            384         PUSH    B                 ; Save present status
0102  F5            385         PUSH    PSW               ;
0103  01FD04        386         LXI     B,1277            ;
                   387 DLL10:                             ;
0106  0B            388         DCX     B                 ; Decrement counter
0107  78            389         MOV     A,B               ;
0108  B1            390         ORA     C                 ;
0109  C20601    C   391         JNZ     DLL10             ; Loop till time up
                   392                                   ;
010C  F1            393         POP     PSW               ; Restore status
010D  C1            394         POP     B                 ;
                   395                                   ;
010E  C9            396         RET                       ;
                   397
                   398 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                399 ;-----------------------------------------------------------------
                400 ;                       Send Message Procedures
                401 ;-----------------------------------------------------------------
                402 ; FUNCTION: SNDMSG
                403 ; INPUTS  : DE pointing at a buffer containing data
                404 ;           Buffer - Length (2 Bytes), DA, [IEEE control byte,]
                405 ;                    Data bytes (x length)
                406 ;           Flags:  A = 0 means send and dont wait for ACK
                407 ;                     = 0FFH means wait for ACK after frame TX
                408 ;                   B = 0 means the frame is a normal UI frame
                409 ;                       0FFH means a control frame
                410 ; OUTPUTS : A = 0 then successful
                411 ;           A = Non-zero then unsuccessful and DE points to the
                412 ;                beginning of the section of the buffer which
                413 ;                failed.  The error codes are:
                414 ;
                415 ;           A = 1 - Insufficient buffers at the receiver
                416 ;               2 - Receiver not enabled
                417 ;               3 - Receiver over-run
                418 ;               4 - Frame exceeded 16 receiver buffers
                419 ;
                420 ;               9 - Transmission failure after retry
                421 ;               A - Transmitter under-run
                422 ;               B - Too little data supplied by host
                423 ;               C - Frame exceeded 16 transmitter buffers
                424 ;
                425 ; DESTROYS: A, B, C, D, E, H, L, Flags
                426 ; DESCRIPTION:
                427 ;           Takes the message from the pointed to by the DE
                428 ;           registers and sends it across the network.  The max
```

```
 LOC   OBJ        LINE      SOURCE STATEMENT

                    429 ;           frame is 311 bytes, and so if the frame is longer
                    430 ;           then SNDMSG will break the contents of the buffer
                    431 ;           into 311 byte blocks and send them.  If any frame
                    432 ;           is unsuccessful on two attempts then the A register
                    433 ;           is set to 1 and the routine returns with the DE
                    434 ;           register pair pointing to the start of the
                    435 ;           unsuccessful block.  If the total transmission is
                    436 ;           successful then A is set to 0 and the routine returns.
                    437 ;           The value of DE is irrelevant in the second case.
                    438 ;           A IEEE control word is added automatically if the
                    439 ;           value of the B register is 0 on entry, if it is 0FFH
                    440 ;           then the control word is expected in the buffer.
                    441 ;
                    442 ;-----------------------------------------------------------------
                    443 SNDMSG:
                    444
                    445 ; Read the flags first
                    446
 010F  3C           447         INR     A               ; Send with ACK?
 0110  CA1801   C   448         JZ      ACKIT           ; Yes
 0113  3E00         449         MVI     A,FALSE         ; No
 0115  C31A01   C   450         JMP     SML1            ;
                    451 ACKIT:                          ;
 0118  3EFF         452         MVI     A,TRUE          ;
                    453 SML1:                           ;
 011A  32470A   C   454         STA     SNDWITHACK      ; Save in the relevant flag
                    455                                 ;
 011D  78           456         MOV     A,B             ; Read the control frame flag
 011E  3C           457         INR     A               ; Control Frame?
 011F  CA2701   C   458         JZ      CTLFRM          ; Yes
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

0122  3E00           459          MVI     A,FALSE          ; No
0124  C32901    C    460          JMP     SML2             ;
                    461 CTLFRM:                           ;
0127  3EFF           462          MVI     A,TRUE           ;
                    463 SML2:                             ;
0129  32480A    C    464          STA     CTLFR            ; Save in the relevant flag
                    465                                   ;
012C  DB70           466          IN      CR0              ; Read the present state
012E  32430A    C    467          STA     PRESSTAT         ;
                    468
                    469 ; Read information from the input buffer
                    470
0131  1A             471          LDAX    D                ; Read Length byte
0132  32450A    C    472          STA     LEN              ;   and save it
0135  13             473          INX     D                ;
0136  1A             474          LDAX    D                ;
0137  32460A    C    475          STA     LEN+1            ;
013A  13             476          INX     D                ; Read Destination Byte
013B  1A             477          LDAX    D                ;
013C  324B0A    C    478          STA     DEST             ;   and save it
013F  13             479          INX     D                ;
0140  EB             480          XCHG                     ;
0141  224C0A    C    481          SHLD    LASTBYTE         ; Save the pointer to the first byte
0144  EB             482          XCHG                     ;   of the new frame
                    483
                    484 ; Read the Buffer to be used
                    485
                    486 SM1:                              ;
0145  3A400A    C    487          LDA     ?TXBUF           ; ?TXBUF contains value of last buf
0148  FE01           488          CPI     1                ;        used- use the other one
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

014A  C2580l   C    489          JNZ     USED2           ;
014D  21FF08   C    490          LXI     H,TXBUF2        ;
0150  3E02          491          MVI     A,2             ;
0152  32400A   C    492          STA     ?TXBUF          ; And save the status
0155  C3600l   C    493          JMP     FILL            ;
                    494 USED2:                           ;
0158  21BF07   C    495          LXI     H,TXBUF1        ;
015B  3E01          496          MVI     A,1             ;
015D  32400A   C    497          STA     ?TXBUF          ;
                    498
                    499 ; Prepare the buffers
                    500
                    501 FILL:                            ;
0160  3600          502          MVI     M,0             ; Clear the links
0162  23            503          INX     H               ;
0163  3600          504          MVI     M,0             ;
0165  23            505          INX     H               ;
0166  3600          506          MVI     M,0             ; Clear FSB
0168  23            507          INX     H               ;
                    508
                    509 ; Calculate the value of the FCB
                    510
0169  3A470A   C    511          LDA     SNDWITHACK      ; ACK or not?
016C  FE00          512          CPI     FALSE           ; No
016E  CA7E01   C    513          JZ      F01             ;
0171  3A4B0A   C    514          LDA     DEST            ; If a broadcast frame dont wait for ack
0174  FEFF          515          CPI     0FFH            ; 0FFH = Broadcast Frame
0176  CA7E01   C    516          JZ      F01             ;
0179  36C0          517          MVI     M,FCBWAIT       ; Set FCB to wait for response
017B  C38001   C    518          JMP     F02             ;
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     519 F01:                                ;
017E   3600          520         MVI     M,FCBNOWT           ; Set the FCB to no waiting for ACK
                     521 F02:                                ;
0180   23            522         INX     H                   ;
                     523
                     524 ; Calculate number of buffers to be used
                     525
0181   D5            526         PUSH    D                   ;
0182   E5            527         PUSH    H                   ;
0183   2A450A    C   528         LHLD    LEN                 ; If LEN < Buffer Then
0186   113601        529         LXI     D,BYTECOUNT-10      ;
0189   CD43F0        530         CALL    DCMP                ;
018C   DAAF01    C   531         JC      LTBUF               ;   Only one buffer
018F   E1            532         POP     H                   ; Retrieve LENGTH field addr
0190   113F01        533         LXI     D,BYTECOUNT-1       ; Write max length to it
0193   72            534         MOV     M,D                 ; First the MSB then the LSB
0194   23            535         INX     H                   ;
0195   73            536         MOV     M,E                 ;
0196   23            537         INX     H                   ;
0197   E5            538         PUSH    H                   ;
0198   2A450A    C   539         LHLD    LEN                 ; Adjust the length buffer
019B   11CAFE        540         LXI     D,-(BYTECOUNT-10)   ;
019E   19            541         DAD     D                   ;
019F   22450A    C   542         SHLD    LEN                 ;
01A2   E1            543         POP     H                   ;
01A3   D1            544         POP     D                   ;
01A4   013601        545         LXI     B,BYTECOUNT-10      ; Set counter
01A7   3EFF          546         MVI     A,TRUE              ; Indicate frame > Buffer
01A9   32440A    C   547         STA     LONG                ;
01AC   C3C701    C   548         JMP     FILDEST             ;
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     549                                  ;
                     550 LTBUF:                           ; Frame Less than one buffer long
 01AF  44            551         MOV     B,H              ; Set up the counter
 01B0  4D            552         MOV     C,L              ;
 01B1  110900        553         LXI     D,9              ; Adjust LEN for LENGTH
 01B4  19            554         DAD     D                ;
 01B5  EB            555         XCHG                     ;
 01B6  210000        556         LXI     H,0              ; Reset LEN
 01B9  22450A    C   557         SHLD    LEN              ;
 01BC  E1            558         POP     H                ;
 01BD  72            559         MOV     M,D              ; Store in LENGTH
 01BE  23            560         INX     H                ;
 01BF  73            561         MOV     M,E              ;
 01C0  23            562         INX     H                ;
 01C1  3E00          563         MVI     A,FALSE          ; Frame < Buffer
 01C3  32440A    C   564         STA     LONG             ;
 01C6  D1            565         POP     D                ;
                     566
                     567 ; Fill in the rest of the buffer
                     568
                     569 FILDEST:
 01C7  3A4B0A    C   570         LDA     DEST             ; Read the Destination Address
 01CA  77            571         MOV     M,A              ;
 01CB  3A420A    C   572         LDA     MYID             ; Read the Source Address
 01CE  23            573         INX     H                ;
 01CF  77            574         MOV     M,A              ;
                     575                                  ;
 01D0  EB            576         XCHG                     ;
 01D1  2A4C0A    C   577         LHLD    LASTBYTE         ; Read addr of last byte
 01D4  224E0A    C   578         SHLD    BEGBUF           ; Save it for the error return
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

01D7  EB             579          XCHG                    ;
                     580                                  ;
01D8  23             581          INX     H               ;
01D9  3A480A   C     582          LDA     CTLFR           ; A control frame?
01DC  FEFF           583          CPI     TRUE            ;
01DE  CAE401   C     584          JZ      COPYIT          ;
01E1  3603           585          MVI     M,UI            ; Supply the control byte
01E3  23             586          INX     H               ; Adjust the pointer
                     587 COPYIT:                          ;
01E4  CDA603   C     588          CALL    COPY            ; Copy (BC) bytes from (DE) to (HL)
01E7  EB             589          XCHG                    ;
01E8  224C0A   C     590          SHLD    LASTBYTE        ; Save the addr of last byte
                     591                                  ;
01EB  21BF07   C     592          LXI     H,TXBUF1        ; Find the last buffer address
01EE  11FF08   C     593          LXI     D,TXBUF2        ;
01F1  3A400A   C     594          LDA     ?TXBUF          ;
01F4  FE02           595          CPI     2               ;
01F6  CAFA01   C     596          JZ      USED1           ;
01F9  EB             597          XCHG                    ;
                     598 USED1:                           ; Link up to the previous buffer
01FA  23             599          INX     H               ;
01FB  73             600          MOV     M,E             ; Least significant byte first
01FC  2B             601          DCX     H               ;
01FD  72             602          MOV     M,D             ;
                     603
                     604 ; Allow to transmit
                     605
01FE  DB70           606          IN      CR0             ; Read present value
0200  F6C0           607          ORI     11000000B       ;
0202  D370           608          OUT     CR0             ; Enable transmission
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     609
                     610 ; Wait for transmission
                     611
0204   EB            612          XCHG                    ;
0205   23            613          INX     H               ;
0206   23            614          INX     H               ; Point at the FSB
                     615 ?SENT:                           ;
0207   7E            616          MOV     A,M             ; Read FSB
0208   47            617          MOV     B,A             ;
0209   E680          618          ANI     10000000B       ; DONE bit set?
020B   C21402    C   619          JNZ     ENDING          ;
020E   CD0101    C   620          CALL    DLAY            ; Wait for WD to update memory
0211   C30702    C   621          JMP     ?SENT           ;
                     622 ENDING:                          ;
0214   78            623          MOV     A,B             ;
0215   E607          624          ANI     07H             ; Was it successful
0217   C22802    C   625          JNZ     END2            ; - No
                     626
                     627 ; Loop if frame > Buffer or else end
                     628
021A   3A440A    C   629          LDA     LONG            ; If LONG = TRUE
021D   FEFF          630          CPI     TRUE            ;   Then Loop
021F   C22802    C   631          JNZ     END2            ;
0222   CD0101    C   632          CALL    DLAY            ;
0225   C34501    C   633          JMP     SM1             ;
                     634 END2:                            ;
0228   2A4E0A    C   635          LHLD    BEGBUF          ; Find beginning of last block sent
022B   EB            636          XCHG                    ;
022C   3A430A    C   637          LDA     PRESSTAT        ; Restore CR0
022F   D370          638          OUT     CR0             ;
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

0231   78           639          MOV     A,B              ;
0232   E60F         640          ANI     0FH              ; Clear top nibble
0234   FE08         641          CPI     08H              ; 8H <> Error
0236   C23B02   C   642          JNZ     END3             ;
0239   3E00         643          MVI     A,0              ; If A = 08H then set to 0
                    644 END3:                            ;
023B   C9           645          RET                      ;
                    646
                    647
                    648 $EJECT
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     649 ;-----------------------------------------------------------
                     650 ;          Procedures for Receiving Frames over the Link
                     651 ;-----------------------------------------------------------
                     652 ; FUNCTION : RECMSG
                     653 ; INPUTS   : None
                     654 ; OUTPUTS  : DE - buffer
                     655 ;             A - 0 if not successful
                     656 ;               - 0FFH if successful
                     657 ; DESCRIPTION:
                     658 ;          The RECMSG procedure will take the frame received from
                     659 ;          network and write it to a buffer in the following format:
                     660 ;            Len ( 2 BYTES), SA , data bytes (x Len)
                     661 ;          The DE registers are then set up to point at the buffer.
                     662 ;
                     663 ;-----------------------------------------------------------
                     664 RECMSG:
023C  DB70           665          IN      CR0              ; Read present status
023E  F660           666          ORI     01100000B        ; Enable receive
0240  D370           667          OUT     CR0              ;
0242  3E2C           668          MVI     A,2CH            ; Set up CR1
0244  D371           669          OUT     CR1              ;
                     670                                   ;
0246  3A3F0A    C    671          LDA     ?RXBUF           ; Find out which buffer was used
0249  FE01           672          CPI     1                ;    last time and use the other
024B  C25C02    C    673          JNZ     RX2              ;
024E  217F06    C    674          LXI     H,RXBUF2         ;
                     675                                   ;
0251  3E02           676          MVI     A,2              ;
0253  32410A    C    677          STA     N?RXBUF          ; Save the new buffer in a temporary
0256  113F05    C    678          LXI     D,RXBUF1         ;     store
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

0259  C36702   C    679          JMP     REC              ;
                   680 RX2:                              ;
025C  213F05   C    681          LXI     H,RXBUF1         ;
025F  3E01          682          MVI     A,1              ;
0261  32410A   C    683          STA     N?RXBUF          ;
0264  117F06   C    684          LXI     D,RXBUF2         ;
                   685
                   686 ; The buffer is now known, see if any data has been received
                   687
                   688 REC:                              ;
0267  23            689          INX     H                ; Point at the FSB
0268  23            690          INX     H                ;
0269  7E            691          MOV     A,M              ;
026A  E680          692          ANI     10000000B        ; If DONE is clear then exit
026C  CA2B03   C    693          JZ      NODATA           ;    as no data yet
                   694
                   695 ; There is data there so receive it
                   696
026F  EB            697          XCHG                     ;
0270  3600          698          MVI     M,0              ; Zero link & FSB in buffer
0272  23            699          INX     H                ;  to be used
0273  3600          700          MVI     M,0              ;
0275  23            701          INX     H                ;
0276  3600          702          MVI     M,0              ;
0278  2B            703          DCX     H                ;
0279  2B            704          DCX     H                ;
027A  EB            705          XCHG                     ; Link present to the next buffer
027B  2B            706          DCX     H                ;
027C  73            707          MOV     M,E              ;
027D  2B            708          DCX     H                ;
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

027E   72            709         MOV     M,D             ;
                     710                                 ;
027F   23            711         INX     H               ; Point at the length bytes
0280   23            712         INX     H               ;
0281   23            713         INX     H               ;
0282   23            714         INX     H               ;
0283   46            715         MOV     B,M             ; Read both length bytes to BC
0284   23            716         INX     H               ;
0285   4E            717         MOV     C,M             ;
0286   EB            718         XCHG                    ;
0287   21F7FF        719         LXI     H,0FFF7H        ; Subtract 9 for the control bytes
028A   09            720         DAD     B               ;    leaving length of data bytes
028B   22F703    C   721         SHLD    MSGBUF          ; Save the length
028E   23            722         INX     H               ;
028F   23            723         INX     H               ;
0290   13            724         INX     D               ;
0291   13            725         INX     D               ;
0292   44            726         MOV     B,H             ; Set BC to the counter value
0293   4D            727         MOV     C,L             ;
0294   C5            728         PUSH    B               ; Save BC for later
0295   010200        729         LXI     B,2             ;
0298   21F903    C   730         LXI     H,MSGBUF+2      ; The start of the message to be here
029B   CDA603    C   731         CALL    COPY            ; Copy (BC) bytes from (DE) to (HL)
029E   C1            732         POP     B               ;
029F   0B            733         DCX     B               ;
02A0   0B            734         DCX     B               ; Adjust the counter
02A1   3AFA03    C   735         LDA     MSGBUF+3        ; Read the control byte
02A4   E610          736         ANI     10H             ; Read the poll bit
02A6   CAAE02    C   737         JZ      NOTPOLL         ;
02A9   3EFF          738         MVI     A,TRUE          ;
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

02AB   C3B002   C    739          JMP     FILPOLL          ;
                     740 NOTPOLL:                          ;
02AE   3E00          741          MVI     A,FALSE          ;
                     742 FILPOLL:                          ;
02B0   32490A   C    743          STA     POLL             ; Set the poll flag
02B3   3E00          744          MVI     A,FALSE          ; Set the default frame flag
02B5   324A0A   C    745          STA     BADFRAME         ;
02B8   3AFA03   C    746          LDA     MSGBUF+3         ;
02BB   E6EF          747          ANI     0EFH             ; Ignore the poll bit now
02BD   FE03          748          CPI     UI               ; UI Frame?
02BF   CAD102   C    749          JZ      GETREST          ;
02C2   FEAF          750          CPI     XID              ; XID Frame?
02C4   CAF802   C    751          JZ      RETXID           ; Return the XID in that case
02C7   FEE3          752          CPI     TEST             ; TEST Frame?
02C9   CA0703   C    753          JZ      RETTEST          ; Return the test frame
02CC   3EFF          754          MVI     A,TRUE           ; Unknown control byte
02CE   324A0A   C    755          STA     BADFRAME         ;
                     756 GETREST:                          ;
02D1   78            757          MOV     A,B              ;
02D2   B1            758          ORA     C                ;
02D3   CA1402   C    759          JZ      ENDING           ; Done
02D6   2B            760          DCX     H                ; Write over the control byte
02D7   CDA603   C    761          CALL    COPY             ;
                     762 ENDIN:                            ;
02DA   3A490A   C    763          LDA     POLL             ; If the poll bit is set then an ERROR
02DD   FEFF          764          CPI     TRUE             ;
02DF   CA2503   C    765          JZ      EREND            ;
02E2   3A4A0A   C    766          LDA     BADFRAME         ;
02E5   FEFF          767          CPI     TRUE             ; Unknown control byte
02E7   CA2503   C    768          JZ      EREND            ;
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

02EA  11F703  C   769          LXI     D,MSGBUF        ; Points at the data
02ED  3A410A  C   770          LDA     N?RXBUF         ; A successful read so store temp
02F0  323F0A  C   771          STA     ?RXBUF          ;    in permanent store
02F3  3EFF        772          MVI     A,0FFH          ; Indicating success
02F5  C33003  C   773          JMP     FINIS           ;
                  774
                  775 ; Now deal with the exceptions
                  776
                  777 RETXID:                           ;
02F8  11EC03  C   778          LXI     D,XIDFRAME      ;
02FB  CD3103  C   779          CALL    DESTPSET        ; Set up the destination address
                  780                                  ;  and set the P/F bit to the received value
02FE  AF          781          XRA     A               ; Send with no ACK
02FF  06FF        782          MVI     B,0FFH          ; Control frame
0301  CD0F01  C   783          CALL    SNDMSG          ; Send the frame
0304  C32503  C   784          JMP     EREND           ; Finish up
                  785                                  ;
                  786 RETTEST:                          ;
0307  2AF703  C   787          LHLD    MSGBUF          ; Read the length word
030A  23          788          INX     H               ; To include the control byte
030B  22F303  C   789          SHLD    TESTFRAME       ;
030E  D5          790          PUSH    D               ; Save present position in Rx buffer
030F  11F303  C   791          LXI     D,TESTFRAME     ;
0312  CD3103  C   792          CALL    DESTPSET        ; Set the Destination address and Poll bit
0315  D1          793          POP     D               ;
0316  21F703  C   794          LXI     H,TESTFRAME+4   ; Data to go there
0319  CDA603  C   795          CALL    COPY            ; Copy the data to the Tx buffer
031C  11F303  C   796          LXI     D,TESTFRAME     ;
031F  AF          797          XRA     A               ; Send with no ACK
0320  06FF        798          MVI     B,0FFH          ; Control frame
```

```
 LOC   OBJ         LINE        SOURCE STATEMENT

0322  CD0F01   C    799          CALL    SNDMSG          ; Send the frame
                   800
                   801 ; Error exit, also used by control frames
                   802
                   803 EREND:                           ;
0325  3A410A   C    804          LDA     N?RXBUF         ; Adjust the pointers
0328  323F0A   C    805          STA     ?RXBUF          ;
                   806
                   807 ; Exit point if no data
                   808
                   809 NODATA:
032B  CD0101   C    810          CALL    DLAY            ; Wait for TAC to update Regs
032E  3E00          811          MVI     A,00            ; Unsuccessful
                   812 FINIS:                           ;
0330  C9            813          RET                     ;
                   814
                   815 $EJECT
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     816 ;-----------------------------------------------------------
                     817 ;                Set the Poll bit and Destination address
                     818 ;-----------------------------------------------------------
                     819 ; FUNCTION: DESTPSET
                     820 ; INPUTS  : DE = Frame to be adjusted
                     821 ;           POLLB = True if F bit to be set
                     822 ;                 = False if F bit to be clear
                     823 ;           MSGBUF+2 = Destination address
                     824 ; OUTPUTS : None
                     825 ; DESTROYS: A, Flags
                     826 ; DESCRIPTION:
                     827 ;                Sets the poll bit to the value it was in the
                     828 ;                received frame, and then sets the destination
                     829 ;                address from MSGBUF.
                     830 ;
                     831 ;-----------------------------------------------------------
                     832 DESTPSET:
0331  D5              833         PUSH    D               ; Save status
0332  C5              834         PUSH    B               ;
0333  3AF903    C     835         LDA     MSGBUF+2        ; Get the destination address
0336  13              836         INX     D               ;
0337  13              837         INX     D               ; Point at Destination address
0338  12              838         STAX    D               ;
0339  13              839         INX     D               ; Point at control byte
033A  3A490A    C     840         LDA     POLL            ; Poll bit set?
033D  FEFF            841         CPI     TRUE            ;
033F  1A              842         LDAX    D               ; Read control byte
0340  C24903    C     843         JNZ     NOSET           ;
0343  0610            844         MVI     B,POLLB         ; Set up the poll bit
0345  B0              845         ORA     B               ;
```

```
 LOC   OBJ         LINE        SOURCE STATEMENT

0346  C34C03   C    846          JMP     POLRET          ; Save it
                   847 NOSET:                            ;
0349  06EF         848          MVI     B,NOPOLL        ;
034B  A0           849          ANA     B               ;
                   850 POLRET:                           ;
034C  12           851          STAX    D               ; Save the value
034D  C1           852          POP     B               ;
034E  D1           853          POP     D               ;
034F  C9           854          RET                     ;
                   855
                   856 $EJECT
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     857 ;-----------------------------------------------------------
                     858 ;                          Status Return
                     859 ;-----------------------------------------------------------
                     860 ; FUNCTION: STAT
                     861 ; INPUTS:   None
                     862 ; OUTPUTS:  DE = buffer containing 16 registers and 11 Event
                     863 ;                    counters
                     864 ; CALLS:    None
                     865 ; DESCRIPTION: This routine returns a copy of the 16 registers
                     866 ;              and the 11 Event counters in a buffer pointed
                     867 ;              to by DE.
                     868 ;-----------------------------------------------------------
                     869 STAT:
0350  11F703   C     870         LXI     D,MSGBUF        ; Get the buffer address
0353  DB70          871         IN      70H             ; Read the Status registers
0355  12            872         STAX    D               ;     into the buffer
0356  13            873         INX     D               ;
0357  DB71          874         IN      71H             ;
0359  12            875         STAX    D               ;
035A  13            876         INX     D               ;
035B  DB72          877         IN      72H             ;
035D  12            878         STAX    D               ;
035E  13            879         INX     D               ;
035F  DB73          880         IN      73H             ;
0361  12            881         STAX    D               ;
0362  13            882         INX     D               ;
0363  DB74          883         IN      74H             ;
0365  12            884         STAX    D               ;
0366  13            885         INX     D               ;
0367  DB75          886         IN      75H             ;
```

```
 LOC   OBJ        LINE       SOURCE STATEMENT

0369   12          887        STAX    D               ;
036A   13          888        INX     D               ;
036B   DB76        889        IN      76H             ;
036D   12          890        STAX    D               ;
036E   13          891        INX     D               ;
036F   DB77        892        IN      77H             ;
0371   12          893        STAX    D               ;
0372   13          894        INX     D               ;
0373   DB78        895        IN      78H             ;
0375   12          896        STAX    D               ;
0376   13          897        INX     D               ;
0377   DB79        898        IN      79H             ;
0379   12          899        STAX    D               ;
037A   13          900        INX     D               ;
037B   DB7A        901        IN      7AH             ;
037D   12          902        STAX    D               ;
037E   13          903        INX     D               ;
037F   DB7B        904        IN      7BH             ;
0381   12          905        STAX    D               ;
0382   13          906        INX     D               ;
0383   DB7C        907        IN      7CH             ;
0385   12          908        STAX    D               ;
0386   13          909        INX     D               ;
0387   DB7D        910        IN      7DH             ;
0389   12          911        STAX    D               ;
038A   13          912        INX     D               ;
038B   DB7E        913        IN      7EH             ;
038D   12          914        STAX    D               ;
038E   13          915        INX     D               ;
038F   DB7F        916        IN      7FH             ;
```

```
 LOC   OBJ         LINE        SOURCE STATEMENT

0391  12            917          STAX    D               ;
0392  13            918          INX     D               ;
                    919                                  ;
0393  213405    C   920          LXI     H,CBLOCK+5      ; Point at the Event counters
0396  060B          921          MVI     B,11            ;
                    922 STALOOP:                         ; Read the 11 Event counters
0398  7E            923          MOV     A,M             ; Read the value
0399  3600          924          MVI     M,0             ; Reset value at same time
039B  12            925          STAX    D               ;
039C  23            926          INX     H               ; Adjust pointers
039D  13            927          INX     D               ;
039E  05            928          DCR     B               ;
039F  C29803    C   929          JNZ     STALOOP         ;
                    930                                  ;
03A2  11F703    C   931          LXI     D,MSGBUF        ; Reset the pointer
03A5  C9            932          RET                     ;
                    933
                    934 $EJECT
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     935 ;-----------------------------------------------------------
                     936 ;                  Copy procedure
                     937 ;-----------------------------------------------------------
                     938 ; FUNCTION: COPY
                     939 ; INPUTS:   BC = counter
                     940 ;           DE = Source address
                     941 ;           HL = Destination address
                     942 ; OUTPUTS:  None
                     943 ; DESTROYS: A, B, C, D, E, H, L, Flags
                     944 ; DESCRIPTION:
                     945 ;       Copies the (BC) bytes of data starting from (DE) and
                     946 ;       writting them to memory starting from (HL).
                     947 ;
                     948 ;-----------------------------------------------------------
                     949 COPY:
03A6  1A             950         LDAX    D               ; Read the data
03A7  77             951         MOV     M,A             ; Store it in destination
03A8  13             952         INX     D               ; Adjust the pointers
03A9  23             953         INX     H               ;
03AA  0B             954         DCX     B               ; End of the data copy?
03AB  78             955         MOV     A,B             ;
03AC  B1             956         ORA     C               ;
03AD  C2A603    C    957         JNZ     COPY            ; NZ = not yet complete so loop
03B0  C9             958         RET                     ;
                     959
                     960 $EJECT
```

```
 LOC   OBJ          LINE        SOURCE STATEMENT

                     961 ;-----------------------------------------------------------
                     962 ;                        Messages
                     963 ;-----------------------------------------------------------
 03B1  4475706C      964 FAILMSG:        DB      'Duplicate TAC addresses',CR,LF,0
       69636174
       65205441
       43206164
       64726573
       7365730D
       0A00
 03CB  54414320      965 READY:          DB      'TAC initialised',CR,LF,0
       696E6974
       69616C69
       7365640D
       0A00
 03DD  54414320      966 OUTITIS:        DB      'TAC isolated',CR,LF,0
       69736F6C
       61746564
       0D0A00
                     967
                     968 ;-----------------------------------------------------------
                     969 ;                      Local Data Segment
                     970 ;-----------------------------------------------------------
                     971 XIDFRAME:                               ; XID response
 03EC  0400          972         DW      4                       ; Length field
 03EE  00            973         DB      0                       ; Destination (to be filled in)
 03EF  AF            974         DB      XID                     ; XID Control byte (adjust F bit = cmnd P bit)
 03F0  810102        975         DB      81H,01H,02H             ; XID I Field = Class I LLC, window size - 1
                     976 ;
                     977 TESTFRAME:                              ; TEST response
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

03F3   0000        978          DW      0              ; Length field (to be filled in)
03F5   00          979          DB      0              ; Destination address (to be filled in)
03F6   E3          980          DB      TEST           ; TEST control byte
                   981                                 ; Use MSGBUF for I field
                   982 ;
                   983 MSGBUF:
03F7               984          DS      312            ; Area for returned data
                   985
                   986 ; DMA Data Buffers
                   987
                   988 CBLOCK:                         ; Control block
052F               989          DS      2              ; Next Receive Buffer (NXTR) (to be filled in)
0531               990          DS      2              ; Next Transmit Buffer (NXTT) (to be filled in)
0533               991          DS      1              ; Buffer Size, 4=320 Bytes
0534               992          DS      11             ; Event Counters
                   993
                   994 RXBUF1:                         ; First receive buffer
053F               995          DS      2              ; Link
0541               996          DS      318            ; Buffer
                   997 RXBUF2:                         ; Second receive buffer
067F               998          DS      2              ; Link
0681               999          DS      318            ; Buffer
                  1000 TXBUF1:                         ; First transmit buffer
07BF              1001          DS      2              ; Link
07C1              1002          DS      318            ; Buffer
                  1003 TXBUF2:                         ; Second transmit buffer
08FF              1004          DS      2              ; Link
0901              1005          DS      318            ; Buffer
                  1006
                  1007 ; General Flag storage
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                  1008
                  1009 ?RXBUF:                          ; Buffer pointers
 0A3F             1010        DS      1                 ;
                  1011 ?TXBUF:                          ;
 0A40             1012        DS      1                 ;
                  1013 N?RXBUF:                         ; Temporary storage
 0A41             1014        DS      1                 ;
                  1015 MYID:                            ; This node's address
 0A42             1016        DS      1                 ;
                  1017 PRESSTAT:                        ; Storage for CR0 status
 0A43             1018        DS      1                 ;
                  1019 LONG:                            ; Tx Frame > 312 Bytes => TRUE
 0A44             1020        DS      1                 ;      Else FALSE
                  1021 LEN:                             ; Bytes left in Tx frame
 0A45             1022        DS      2                 ;
                  1023 SNDWITHACK:                      ; Acknowledge flag
 0A47             1024        DS      1                 ;
                  1025 CTLFR:                           ; Control frame flag
 0A48             1026        DS      1                 ;
                  1027 POLL:                            ; Poll bit flag
 0A49             1028        DS      1                 ;
                  1029 BADFRAME:                        ; Bad frame control word
 0A4A             1030        DS      1                 ;
                  1031 DEST:                            ; Tx frame Destination ID
 0A4B             1032        DS      1                 ;
                  1033 LASTBYTE:                        ; Store last byte read
 0A4C             1034        DS      2                 ;
                  1035 BEGBUF:                          ; Store for Beginning of block
 0A4E             1036        DS      2                 ;    pointer
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                  1037 ADDUP:                          ; Address duplication counter
 0A50             1038         DS      1              ;
                  1039
                  1040 ; Stack Area
                  1041
 0A51             1042         DS      30             ;
                  1043 NEWSP:                          ;
                  1044         END
```

0 error(s) detected

LOC OBJ LINE SOURCE STATEMENT

User symbols

?RXBUF	0A3F	C	?SENT	0207	C	?TXBUF	0A40	C	ACKIT	0118	C	ADDUP	0A50	C	AHOLT	007D	A
BADFRA	0A4A	C	BEGBUF	0A4E	C	BUFLEN	0004	A	BYTECO	0140	A	C28253	000A	A	CBLOCK	052F	C
CBPH	007A	A	CBPL	007B	A	CI	F013	A	CITEST	F00A	A	CONT82	000B	A	CONTWD	00B0	A
COPY	03A6	C	COPYIT	01E4	C	CR	000D	A	CR0	0070	A	CR1	0071	A	CTLFR	0A48	C
CTLFRM	0127	C	CTR0	0076	A	CTRLZ	001A	A	CYCLES	0000	A	DCMP	F043	A	DEAD	00A0	C
DEST	0A4B	C	DESTPS	0331	C	DIAG1	007F	C	DLAY	0101	C	DLL10	0106	C	DWNITI	00F9	C
END2	0228	C	END3	023B	C	ENDIN	02DA	C	ENDING	0214	C	EREND	0325	C	F01	017E	C
F02	0180	C	FAILMS	03B1	C	FAILSD	00DD	C	FALSE	0000	A	FCBNOW	0000	A	FCBWAI	00C0	A
FDIAG	0091	C	FILDES	01C7	C	FILL	0160	C	FILPOL	02B0	C	FINIS	0330	C	GETRES	02D1	C
IR0	0073	A	LASTBY	0A4C	C	LEN	0A45	C	LF	000A	A	LONG	0A44	C	LOOPD	0062	C
LTBUF	01AF	C	MA	007F	A	MAXFRA	0001	A	MSGBUF	03F7	C	MYID	0A42	C	N?RXBU	0A41	C
NA	0077	A	NAR	007C	A	NETCTR	0070	A	NEWSP	0A6F	C	NODATA	032B	C	NONDEA	006F	C
NOPOLL	00EF	A	NOSET	0349	C	NOTPOL	02AE	C	NTWRKI	000F	C	OUTITI	03DD	C	OUTMSG	F03D	A
POLL	0A49	C	POLLB	0010	A	POLRET	034C	C	PRESST	0A43	C	READY	03CB	C	READY1	00AA	C
REC	0267	C	RECMSG	023C	C	RESPON	0010	A	RETTES	0307	C	RETXID	02F8	C	RX2	025C	C
RXBUF1	053F	C	RXBUF2	067F	C	SM1	0145	C	SML1	011A	C	SML2	0129	C	SNDMSG	010F	C
SNDWIT	0A47	C	SR0	0072	A	SR1	0074	A	SR2	0075	A	STALOO	0398	C	STAT	0350	C
STCHG	0057	C	STNDWN	00E3	C	TA	0078	A	TD	0079	A	TEST	00E3	A	TESTFR	03F3	C
THISNO	0002	A	TIMEOU	0080	A	TRUE	FFFF	A	TXBUF1	07BF	C	TXBUF2	08FF	C	TXLT	007E	A
UI	0003	A	UIFR	0003	A	USED1	01FA	C	USED2	0158	C	WAITON	00E9	C	XID	00AF	A
XIDFRA	03EC	C															

APPENDIX F

SNIOS for the TAC Network

```
 LOC  OBJ        LINE        SOURCE STATEMENT

                    1 $PAGELENGTH(35)
                    2 $TITLE('TAC SLAVE NETWORK I/O SYSTEM')
                    3 ;*****************************************************************
                    4 ;*****************************************************************
                    5 ;**                                                             **
                    6 ;**      S l a v e   N e t w o r k   I / O   S y s t e m        **
                    7 ;**                                                             **
                    8 ;**            F o r   T o k e n   P a s s i n g                **
                    9 ;**                                                             **
                   10 ;**           N e t w o r k   C o n t r o l l e r              **
                   11 ;**                                                             **
                   12 ;*****************************************************************
                   13 ;*****************************************************************
                   14 ;
                   15 ;       PROGRAM        : TOKSNIOS.ASM
                   16 ;       LANGUAGE       : 8085 Assembler Code
                   17 ;       TARGET SYSTEM  : UCT SABus Kits
                   18 ;       AUTHOR         : Q.P. Mc Grath
                   19 ;       DATE           : 13 August 1985
                   20 ;
                   21 ; TOKSNIOS is the user written program which forms the link
                   22 ; between the local CP/M system and the network. The data is
                   23 ; transmitted, under this programs control, across the network
                   24 ; and to and from the the Network I/O System (NIOS), which is the
                   25 ; link to the local CP/M system.
                   26 ;
                   27 ; Once initialisation is complete, TOKSNIOS awaits data from the
                   28 ; NIOS. The NIOS fills the buffer with data in the format below
                   29 ; and then sets up the DE register pair to point at the buffer.
                   30 ; The data format to and from the NIOS is:
```

```
 LOC  OBJ        LINE        SOURCE STATEMENT

                    31 ;
                    32 ;       FMT - The format code. For CP/NET 1.2 there is only one
                    33 ;             possible format pair, 00H for transmission of data
                    34 ;             and 01H for returned data. This allows 1 byte for
                    35 ;             each of the Destination ID code, the Source ID code
                    36 ;             Function, Size field, and 256 bytes for the data
                    37 ;             field.
                    38 ;       DID - The Destination ID code field.
                    39 ;       SID - Source ID code field.
                    40 ;       FNC - Function code field.
                    41 ;       SIZ - Size -1 field.
                    42 ;       DB0 - First data byte field
                    43 ;       DB1 - Second data byte field
                    44 ;        !              !
                    45 ;       DBn - Nth data byte field. (to a maximum of 256 bytes)
                    46 ;
                    47 ; The above data is then adjusted slightly to fit into the format
                    48 ; required by the TAC User Routines (TACUR) as shown below, and
                    49 ; transmitted across the network using the TACUR program:
                    50 ;
                    51 ;       LENGTH (H) - The LSB length byte. (calculated from the SIZ byte)
                    52 ;       LENGTH (L) - The MSB length byte.
                    53 ;       DA         - The destination ID code which exactly the same as
                    54 ;                    the DID above.
                    55 ;       Data       - The data field. In this section the FNC and SIZ
                    56 ;                    bytes are added to the data bytes.
                    57 ;
                    58 ; The reverse occurs with the incomming data, where it is converted
                    59 ; from the TAC format to the NIOS format. (The FMT code is assumed
                    60 ; to be 00H or 01H and is not transferred across the network.)
```

```
 LOC  OBJ        LINE        SOURCE STATEMENT

                   61 ;
                   62 ; The Requester is always the initiator in the transmission of
                   63 ; data and so the sequence SEND data RECEIVE data is always
                   64 ; followed.
                   65 ;
                   66 ;-----------------------------------------------------------------
                   67 $EJECT
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                    68 ;=============================================================
                    69 ;
                    70 ;                          EQUATES
                    71 ;
                    72 ;=============================================================
                    73
                    74 ;-------------------------------------------------------------
                    75 ;                          True or False Equates
                    76 ;-------------------------------------------------------------
                    77
0000                78 FALSE    EQU     0
FFFF                79 TRUE     EQU     NOT FALSE
                    80
0001                81 SERVERID EQU     01H
007F                82 MA       EQU     7FH
                    83
                    84 ;-------------------------------------------------------------
                    85 ;                          Network Status Byte Equates
                    86 ;-------------------------------------------------------------
                    87
0010                88 ACTIVE           EQU     00010000B       ; slave logged in
0002                89 RCVERR           EQU     00000010B       ; error in received msg
0001                90 SNDERR           EQU     00000001B       ; unable to send message
                    91
                    92 ;-------------------------------------------------------------
                    93 ;                          General Equates
                    94 ;-------------------------------------------------------------
                    95
000A                96 LF       EQU     0AH     ; Line Feed
000D                97 CR       EQU     0DH     ; Carriage Return
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                    98
0064                99 RETRYCNT EQU    100     ; Retry counter for receive message
00CC               100 DLCOUNT EQU     204     ; Counter for the delay loop
                   101
                   102 ;-----------------------------------------------------------
                   103 ;                        BDOS Call equates
                   104 ;-----------------------------------------------------------
                   105
0005               106 BDOS            EQU  0005H     ; BDOS call address
                   107
0040               108 LOGIN           EQU  64        ; Login NDOS function
0009               109 PRINTSTRING     EQU  9         ; Bdos call for string print
                   110
                   111 ;-----------------------------------------------------------
                   112 ;                        TACUR Equates
                   113 ;-----------------------------------------------------------
                   114
D000               115 NETINIT EQU     0D000H          ; TAC and network init routine
D003               116 SNDMSG  EQU     0D003H          ; Transmission routine
D006               117 RECMSG  EQU     0D006H          ; Receive routine
                   118
                   119 $EJECT
```

```
 LOC   OBJ        LINE      SOURCE STATEMENT

                   120 ;=================================================================
                   121 ;                Jump vector for SNIOS entry points
                   122 ;=================================================================
                   123
0000  C38E02       124          JMP     NTWRKINIT       ; network initialization
0003  C38103       125          JMP     NTWRKSTS        ; network status
0006  C38C03       126          JMP     CNFGTBLADR      ; return config table addr
0009  C3B302       127          JMP     SENDMSG         ; send message on network
000C  C31603       128          JMP     RECEIVEMSG      ; receive message from network
000F  C36803       129          JMP     NTWRKERROR      ; network error
0012  C36903       130          JMP     NTWRKWBOOT      ; network warm boot
                   131
                   132 ;=================================================================
                   133 ;
                   134 ;                        Local Data Segment
                   135 ;
                   136 ;=================================================================
                   137
                   138 ;-----------------------------------------------------------------
                   139 ;                     Slave Configuration Table
                   140 ;-----------------------------------------------------------------
                   141 CONFIGTBL:
                   142 NETWORKSTATUS:
0015  00           143          DB      00000000B       ; network status byte
0016                144 SLVID:   DS      1               ; slave processor ID number
0017  0000         145          DB      0,0             ; A:  Disk device
0019  0000         146          DB      0,0             ; B:   "
001B  0000         147          DB      0,0             ; C:   "
001D  0000         148          DB      0,0             ; D:   "
001F  0000         149          DB      0,0             ; E:   "
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

0021   0000        150          DB      0,0              ; F:    "
0023   0000        151          DB      0,0              ; G:    "
0025   0000        152          DB      0,0              ; H:    "
0027   0000        153          DB      0,0              ; I:    "
0029   0000        154          DB      0,0              ; J:    "
002B   0000        155          DB      0,0              ; K:    "
002D   0000        156          DB      0,0              ; L:    "
002F   0000        157          DB      0,0              ; M:    "
0031   0000        158          DB      0,0              ; N:    "
0033   0000        159          DB      0,0              ; O:    "
0035   0000        160          DB      0,0              ; P:    "
                   161
0037   0000        162          DB      0,0              ; console device
                   163
0039   0000        164          DB      0,0              ; list device:
003B   00          165          DB      0                ;                buffer index
003C   00          166          DB      0        ; FMT
003D   01          167          DB      SERVERID ; DID
003E               168 SLVID1:  DS      1        ; SID
003F   05          169          DB      5        ; FNC
0040   00          170          DB      0        ; SIZ
0041               171          DS      1        ; MSG(0)  List number
0042               172          DS      255      ; MSG(1) ... MSG(255)
                   173
                   174 ;-----------------------------------------------------------
                   175 ;                        Data Buffers
                   176 ;-----------------------------------------------------------
                   177 DBUFF:
0141               178          DS      320              ; Buffer
                   179
```

```
 LOC   OBJ         LINE        SOURCE STATEMENT

                    180 ;-----------------------------------------------------------
                    181 ;               Storage for Message address from NDOS
                    182 ;-----------------------------------------------------------
                    183 MSGADR:
 0281  0000        184         DW      0               ; message address
                    185 SLAVEID:
 0283               186         DS      1               ; Save for My Address
                    187
                    188 ;-----------------------------------------------------------
                    189 ;                          Warm Boot Message
                    190 ;-----------------------------------------------------------
                    191 WBOOTMSG:
 0284  0D3C4350    192         DB      CR,'<CP/NET>'
       2F4E4554
       3E
 028D  24           193         DB      '$'
                    194
                    195 $EJECT
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                   196 ;=============================================================
                   197 ;
                   198 ;                  NETWORK INITIALISATION
                   199 ;
                   200 ;=============================================================
                   201 ;
                   202 ; NAME         : NTWRKINIT
                   203 ; INPUTS       : None
                   204 ; OUTPUTS      : A - containing 0 if initialisation was successful
                   205 ;                  - containing 0FFH if unsuccessful
                   206 ; CALLS        : External NETINIT
                   207 ; DESTROYS     : None
                   208 ; DESCRIPTION  : NTWRKINIT sets up the TAC's parameters and then
                   209 ;                  checks the network for activity. If activity is
                   210 ;                  found then it will check for duplicate addresses
                   211 ;                  if found it will tell the user and return with A
                   212 ;                  set to FFH, if no duplication of addresses occurs
                   213 ;                  then it will enable transmission. Should the
                   214 ;                  network be found to be dead, then it will wait for
                   215 ;                  activity and then check for duplicate addresses.
                   216 ;                  In the event of no duplicated addresses the TAC is
                   217 ;                  enabled for transmission and the A register is set
                   218 ;                  to 0H.
                   219 ;
                   220 ;-------------------------------------------------------------
                   221 NTWRKINIT:
                   222
 028E  C5          223         PUSH    B               ; Save registers
 028F  D5          224         PUSH    D               ;
 0290  E5          225         PUSH    H               ;
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

0291  CD00D0       226         CALL    NETINIT          ; Call the network init program in TACUR
0294  D29C02       227         JNC     TACOK            ; If no carry then init was successful
0297  3EFF         228         MVI     A,0FFH           ; Else there was an error
0299  C3AF02       229         JMP     ENDIT            ;
                   230
                   231 ; This means a successful initialisation
                   232
                   233 TACOK:                           ;
029C  DB7F         234         IN      MA               ; Read my address
029E  321600       235         STA     SLVID            ; Store in frame
02A1  323E00       236         STA     SLVID1           ;
02A4  328302       237         STA     SLAVEID          ; For later
02A7  3E10         238         MVI     A,ACTIVE         ; Set the network status byte
02A9  211500       239         LXI     H,NETWORKSTATUS  ;
02AC  B6           240         ORA     M                ;
02AD  77           241         MOV     M,A              ;
02AE  AF           242         XRA     A                ; Return code is 0=success
                   243 ENDIT:                           ;
02AF  E1           244         POP     H                ; Restore registers
02B0  D1           245         POP     D                ;
02B1  C1           246         POP     B                ;
02B2  C9           247         RET                      ;
                   248
                   249
                   250 $EJECT
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                   251 ;=============================================================
                   252 ;
                   253 ;                  SEND MESSAGE ON THE NETWORK
                   254 ;
                   255 ;=============================================================
                   256 ;
                   257 ; NAME         : SENDMSG
                   258 ; INPUTS       : BC containing the message address
                   259 ; OUTPUTS      : Frame to the network
                   260 ; CALLS        : TACUR SNDMSG
                   261 ; DESTROYS     : B, C, H, L
                   262 ; DESCRIPTION : SendMsg takes the message that is written in the
                   263 ;                 address given by BC and writes it into the
                   264 ;                 form required by TACUR. Control is then given
                   265 ;                 to TACUR which sends the frame.
                   266 ;
                   267 ;-------------------------------------------------------------
                   268 SENDMSG:
 02B3  60          269          MOV     H,B              ;
 02B4  69          270          MOV     L,C              ; HL = message address
 02B5  2281Ø2      271          SHLD    MSGADR           ; Save the message address
 02B8  C5          272          PUSH    B                ;
                   273 SML1Ø:                            ;
 02B9  CD06DØ      274          CALL    RECMSG           ; Flush the receive buffer
 02BC  B7          275          ORA     A                ; Ø = flush complete as no data read
 02BD  C2B902      276          JNZ     SML1Ø            ; Or loop
                   277                                   ;
 02CØ  C1          278          POP     B                ;
 02C1  Ø3          279          INX     B                ; Point at the size field
 02C2  Ø3          280          INX     B                ;
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

02C3   03          281          INX     B                ;
02C4   03          282          INX     B                ;
02C5   0A          283          LDAX    B                ;
02C6   6F          284          MOV     L,A              ; Add on the extra control fields
02C7   2600        285          MVI     H,0              ;
02C9   23          286          INX     H                ;
02CA   23          287          INX     H                ;
02CB   23          288          INX     H                ; (HL) = Length plus the control bytes
02CC   2241 01     289          SHLD    DBUFF            ;
02CF   0B          290          DCX     B                ;
02D0   0B          291          DCX     B                ;
02D1   0B          292          DCX     B                ; BC point at DID in the NDOS frame
02D2   1143 01     293          LXI     D,DBUFF+2        ; Start of the data buffer
                   294
                   295 ; Copy the data into the transmission buffer
                   296
02D5   0A          297          LDAX    B                ; Read desination address
02D6   12          298          STAX    D                ;
02D7   03          299          INX     B                ; Read past source ID
02D8   03          300          INX     B                ;
02D9   13          301          INX     D                ;
02DA   2B          302          DCX     H                ;
02DB   7C          303          MOV     A,H              ; End yet?
02DC   B5          304          ORA     L                ;
02DD   CAEA02      305          JZ      DLEND            ;
                   306 DLOOP:                            ;
02E0   0A          307          LDAX    B                ; Read character
02E1   12          308          STAX    D                ; Store it
02E2   03          309          INX     B                ;
02E3   13          310          INX     D                ;
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

 02E4  2B          311          DCX     H                ; Length at an end?
 02E5  7C          312          MOV     A,H              ;
 02E6  B5          313          ORA     L                ;
 02E7  C2E002      314          JNZ     DLOOP            ;
                   315
                   316 ; Now send it using the TACUR SNDMSG
                   317
                   318 DLEND:                            ;
 02EA  114101      319          LXI     D,DBUFF          ; Set up pointer
 02ED  3EFF        320          MVI     A,0FFH           ; Set up flags
 02EF  0600        321          MVI     B,0              ;   for send with ACKs and not control
 02F1  CD03D0      322          CALL    SNDMSG           ;
 02F4  FE00        323          CPI     0                ; Successful?
 02F6  CA0503      324          JZ      TXSUC            ;
 02F9  3E01        325          MVI     A,SNDERR         ; Set the status byte
 02FB  211500      326          LXI     H,NETWORKSTATUS  ;
 02FE  B6          327          ORA     M                ;
 02FF  77          328          MOV     M,A              ;
 0300  3EFF        329          MVI     A,0FFH           ; No - return 0FFH as indication
 0302  C31003      330          JMP     TXS1             ;
                   331 TXSUC:                            ;
 0305  3E01        332          MVI     A,SNDERR         ; Set the status byte
 0307  EEFF        333          XRI     0FFH             ;
 0309  211500      334          LXI     H,NETWORKSTATUS  ;
 030C  A6          335          ANA     M                ;
 030D  77          336          MOV     M,A              ;
 030E  3E00        337          MVI     A,0              ;
                   338 TXS1:                             ;
 0310  2A8102      339          LHLD    MSGADR           ; Restore the message address
 0313  44          340          MOV     B,H              ;
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

0314   4D          341         MOV     C,L             ;
0315   C9          342         RET
                   343
                   344
                   345 $EJECT
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                   346 ;=============================================================
                   347 ;
                   348 ;                 RECEIVE A MESSAGE FROM THE NETWORK
                   349 ;
                   350 ;=============================================================
                   351 ;
                   352 ; NAME         : RECEIVEMSG
                   353 ; INPUTS       : BC containing the address in which to place message
                   354 ; OUTPUTS      : Message to the buffer given
                   355 ; CALLS        : External TACUR RECMSG
                   356 ; DESTROYS     : A, B, C, H, L, Flags
                   357 ; DESCRIPTION : ReceiveMsg polls the TACUR RECMSG for an input and
                   358 ;                 then  places the frame in the address specified by
                   359 ;                 the address in BC.
                   360 ;
                   361 ;-------------------------------------------------------------
                   362 RECEIVEMSG:
 0316  60          363         MOV     H,B             ;
 0317  69          364         MOV     L,C             ;
 0318  228102      365         SHLD    MSGADR          ; Store for return
                   366
                   367 ; Wait for a frame
                   368
 031B  216400      369         LXI     H,RETRYCNT      ;
                   370 RXLOOP:                         ;
 031E  CD06D0      371         CALL    RECMSG          ; RECMSG returns 0 if no frame
 0321  FE00        372         CPI     0               ;
 0323  C22F03      373         JNZ     RXL01           ; <>0 means frame received
 0326  2B          374         DCX     H               ; Check on the retry count
 0327  7C          375         MOV     A,H             ;
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

0328   B5          376          ORA     L               ;
0329   C21E03      377          JNZ     RXLOOP          ;
032C   C35B03      378          JMP     ERREXIT         ; If retry count runs out = error
                   379
                   380 ; A frame was received
                   381
                   382 RXL01:                           ;
032F   1A          383          LDAX    D               ; Read length
0330   6F          384          MOV     L,A             ;
0331   13          385          INX     D               ;
0332   1A          386          LDAX    D               ;
0333   13          387          INX     D               ;
0334   67          388          MOV     H,A             ; (HL) = length of frame
0335   23          389          INX     H               ;
                   390
                   391 ; Now copy the frame to the correct place in memory
                   392
0336   3E01        393          MVI     A,01H           ; Set up the FORMAT byte
0338   02          394          STAX    B               ;
0339   03          395          INX     B               ;
033A   3A8302      396          LDA     SLAVEID         ; Write in DA
033D   02          397          STAX    B               ;
033E   13          398          INX     D               ; To ignore SA given as there is one
033F   03          399          INX     B               ;    in data field
                   400 RXL10:                           ;
0340   1A          401          LDAX    D               ; Write data
0341   02          402          STAX    B               ;
0342   13          403          INX     D               ;
0343   03          404          INX     B               ;
0344   2B          405          DCX     H               ; End of Data?
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

 0345  7C          406          MOV     A,H              ;
 0346  B5          407          ORA     L                ;
 0347  C24003      408          JNZ     RXL10            ;
                   409                                   ;
 034A  3E02        410          MVI     A,RCVERR         ; Set the status byte
 034C  EEFF        411          XRI     0FFH             ;
 034E  211500      412          LXI     H,NETWORKSTATUS  ;
 0351  A6          413          ANA     M                ;
 0352  77          414          MOV     M,A              ;
 0353  3E00        415          MVI     A,00H            ; An indicator to NDOS
 0355  2A8102      416          LHLD    MSGADR           ; Restore the address of the message
 0358  C36703      417          JMP     ENDING           ;
                   418 ERREXIT:                          ;
 035B  3E02        419          MVI     A,RCVERR         ; Set up the status bit
 035D  211500      420          LXI     H,NETWORKSTATUS  ;
 0360  B6          421          ORA     M                ;
 0361  77          422          MOV     M,A              ;
 0362  CD6803      423          CALL    NTWRKERROR       ; Do the required adjustments for hardware
 0365  3EFF        424          MVI     A,0FFH           ; As a flag
                   425 ENDING:                           ;
 0367  C9          426          RET                      ;
                   427
                   428 $EJECT
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                   429 ;=============================================================
                   430 ;
                   431 ;                  GENERAL UTILITIES
                   432 ;
                   433 ;=============================================================
                   434
                   435 ;=============================================================
                   436 ;
                   437 ;                  ERROR CONDITION ROUTINE
                   438 ;
                   439 ;=============================================================
                   440 ;
                   441 ; NAME         : NTWRKERROR
                   442 ; INPUTS       : None
                   443 ; OUTPUTS      : None
                   444 ; CALLS        : None
                   445 ; DESTROYS     : None
                   446 ; DESCRIPTION  : NTWRKERROR does nothing at this stage, but is
                   447 ;                 called every time a network error occurs.
                   448 ;
                   449 ;-------------------------------------------------------------
                   450 NTWRKERROR:
                   451           ;  perform any required device re-initialization
 0368  C9          452           RET
                   453
                   454 $EJECT
```

```
 LOC   OBJ        LINE      SOURCE STATEMENT

                   455 ;=================================================================
                   456 ;
                   457 ;                  NETWORK REBOOT
                   458 ;
                   459 ;=================================================================
                   460 ;
                   461 ; NAME         : NTWRKWBOOT
                   462 ; INPUTS       : None
                   463 ; OUTPUTS      : None
                   464 ; CALLS        : BDOS
                   465 ; DESTROYS     : A, C, Flags
                   466 ; DESCRIPTION  : NTWRKWBOOT is another routine which is empty at
                   467 ;                 this stage apart from the boot up message.
                   468 ;                 This procedure is called each time the CCP is
                   469 ;                 reloaded from disk.  At this point you could
                   470 ;                 possibly check for any mail at the server or
                   471 ;                 do any other task.
                   472 ;
                   473 ;-----------------------------------------------------------------
                   474 NTWRKWBOOT:
0369  0E09          475         MVI     C,9
036B  118402        476         LXI     D,WBOOTMSG
036E  CD0500        477         CALL    BDOS
0371  C9            478         RET
                   479
                   480 $EJECT
```

```
 LOC   OBJ        LINE      SOURCE STATEMENT

                   481 ;=================================================================
                   482 ;
                   483 ;                  DELAY ROUTINE
                   484 ;
                   485 ;=================================================================
                   486 ;
                   487 ; NAME         : DLAY
                   488 ; INPUTS       : None
                   489 ; OUTPUTS      : None
                   490 ; CALLS        : None
                   491 ; DESTROYS     : None
                   492 ; DESCRIPTION  : DLAY creates a delay related to DLCOUNT
                   493 ;
                   494 ;-----------------------------------------------------------------
                   495 DLAY:
0372  F5            496          PUSH    PSW              ; Save all the registers
0373  C5            497          PUSH    B                ;
0374  0ECC          498          MVI     C,DLCOUNT        ; Create the correct time delay
                   499 DLAY1:
0376  E3            500          XTHL                     ; To create some type of delay
0377  E3            501          XTHL                     ;
0378  E3            502          XTHL                     ;
0379  E3            503          XTHL                     ;
037A  0D            504          DCR     C                ; End of the time?
037B  C27603        505          JNZ     DLAY1            ;
                    506
037E  C1            507          POP     B                ; Restore the Registers
037F  F1            508          POP     PSW              ;
0380  C9            509          RET
                    510
```

```
  LOC    OBJ          LINE          SOURCE STATEMENT

                       511 $EJECT
```

```
 LOC   OBJ        LINE        SOURCE STATEMENT

                   512 ;=============================================================
                   513 ;
                   514 ;                  NETWORK STATUS ROUTINE
                   515 ;
                   516 ;=============================================================
                   517 ;
                   518 ; NAME         : NTWRKSTS
                   519 ; INPUTS       : None
                   520 ; OUTPUTS      : Network status to memory NETWORKSTATUS
                   521 ; CALLS        : None
                   522 ; DESTROYS     : B, Flags
                   523 ; DESCRIPTION  : NTWRKSTS reads the old network status and then
                   524 ;                 adjusts it to include the new status. This update
                   525 ;                 occurs at regular intervals.
                   526 ;
                   527 ;-------------------------------------------------------------
                   528 NTWRKSTS:
0381  3A1500       529          LDA     NETWORKSTATUS
0384  47           530          MOV     B,A
0385  E6FC         531          ANI     NOT (RCVERR+SNDERR)
0387  321500       532          STA     NETWORKSTATUS
038A  78           533          MOV     A,B
038B  C9           534          RET
                   535
                   536 $EJECT
```

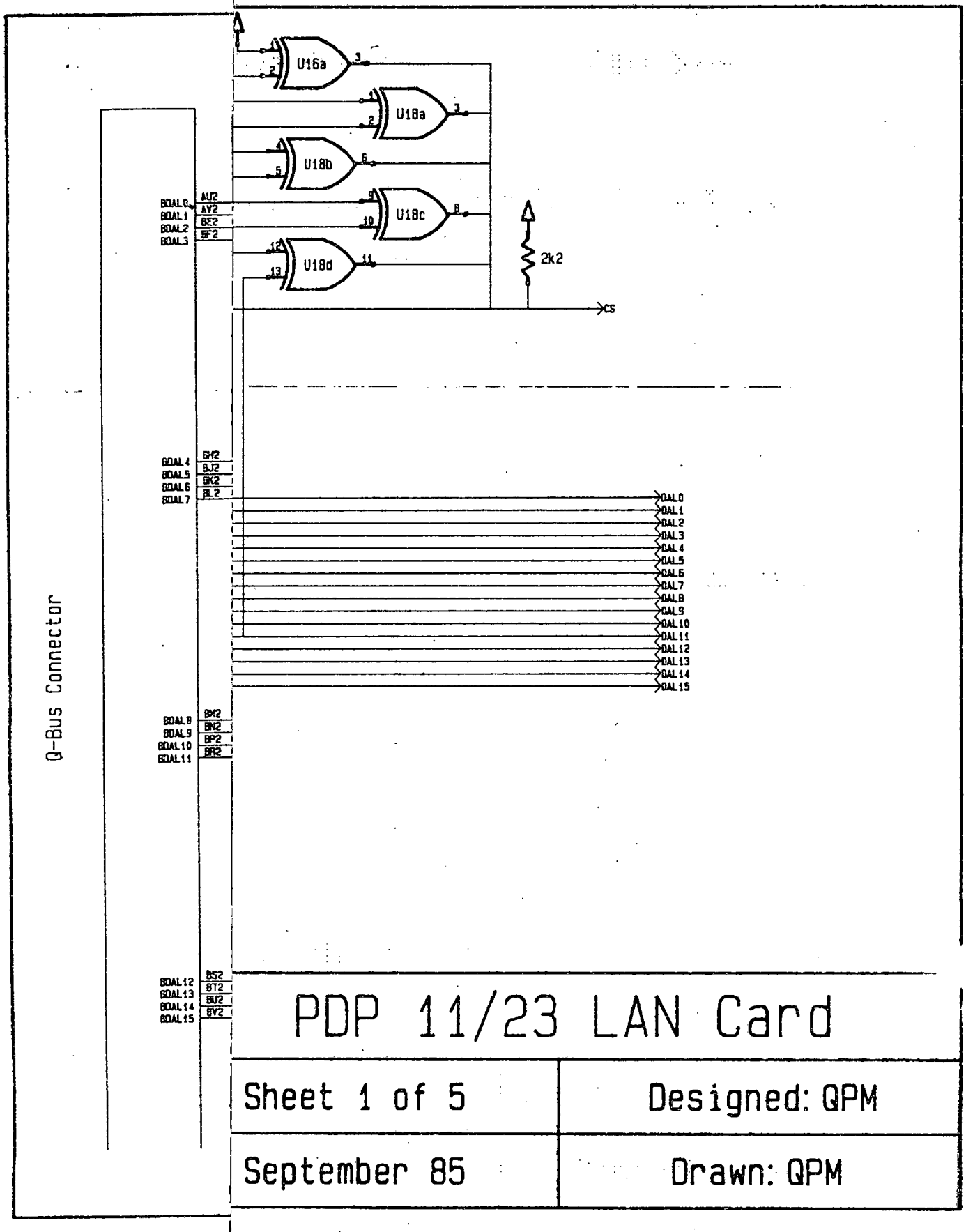
```
  LOC   OBJ        LINE        SOURCE STATEMENT

                    537 ;=============================================================
                    538 ;
                    539 ;                 CONFIGURATION TABLE ADDRESS
                    540 ;
                    541 ;=============================================================
                    542 ;
                    543 ; NAME         : CNFGTBLADR
                    544 ; INPUTS       : None
                    545 ; OUTPUTS      : HL - containing the configuration table address
                    546 ; CALLS        : None
                    547 ; DESTROYS     : H, L
                    548 ; DESCRIPTION : CNFGTBLADR simply loads the address of the
                    549 ;                 requesters configuration table into the HL
                    550 ;                 register pair.
                    551 ;
                    552 ;-------------------------------------------------------------
                    553 CNFGTBLADR:
 038C  211500       554         LXI     H,CONFIGTBL
 038F  C9           555         RET
                    556
                    557
                    558         END

0 error(s) detected


User symbols
```

LOC OBJ LINE SOURCE STATEMENT

ACTIVE	0010	A	BDOS	0005	A	CNFGTB	038C	A	CONFIG	0015	A	CR	000D	A	DBUFF	0141	A	DLAY	0372	A
DLAY1	0376	A	DLCOUN	00CC	A	DLEND	02EA	A	DLOOP	02E0	u	ENDING	0367	A	ENDIT	02AF	A	ERREXI	035B	A
FALSE	0000	A	LF	000A	A	LOGIN	0040	A	MA	007F	A	MSGADR	0281	A	NETINI	D000	A	NETWOR	0015	A
NTWRKE	0368	A	NTWRKI	028E	A	NTWRKS	0381	A	NTWRKW	0369	A	PRINTS	0009	A	RCVERR	0002	A	RECEIV	0316	A
RECMSG	D006	A	RETRYC	0064	A	RXL01	032F	A	RXL10	0340	A	RXLOOP	031E	A	SENDMS	02B3	A	SERVER	0001	A
SLAVEI	0283	A	SLVID	0016	A	SLVID1	003E	A	SML10	02B9	A	SNDERR	0001	A	SNDMSG	D003	A	TACOK	029C	A
TRUE	FFFF	A	TXS1	0310	A	TXSUC	0305	A	WBOOTM	0284	A									

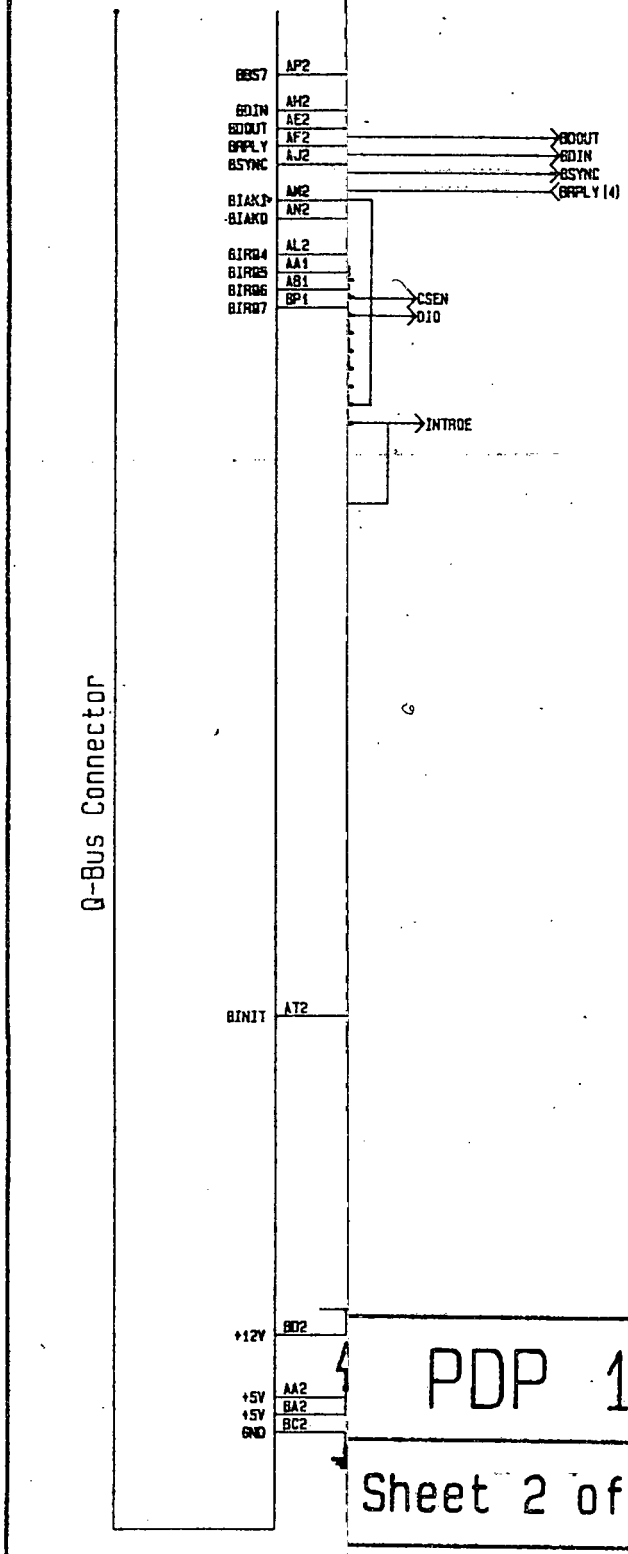
APPENDIX G

The PDP 11/23 LAN Card Circuit Diagrams



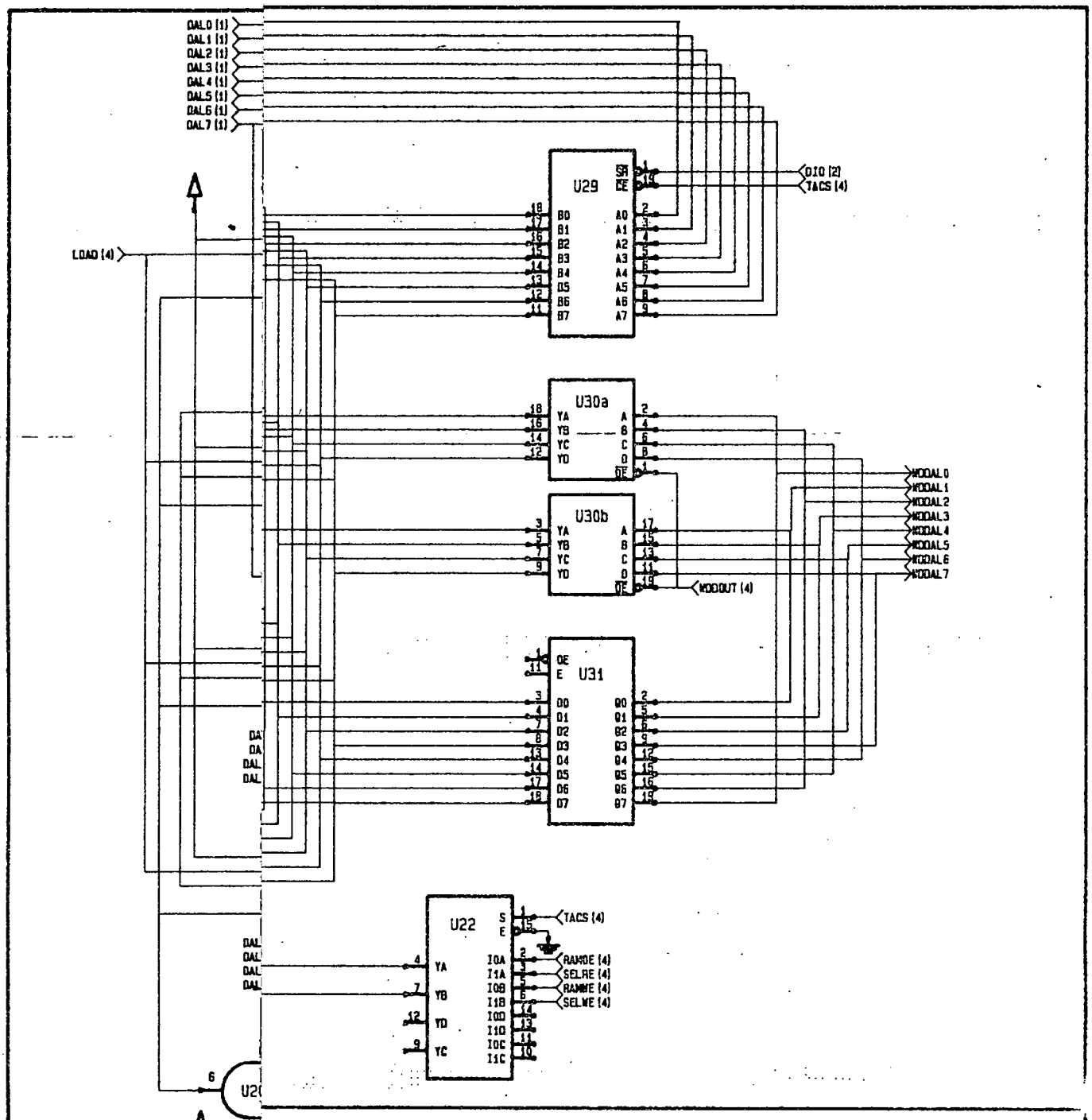
PDP 11/23 LAN Card

Sheet 1 of 5	Designed: QPM
September 85	Drawn: QPM



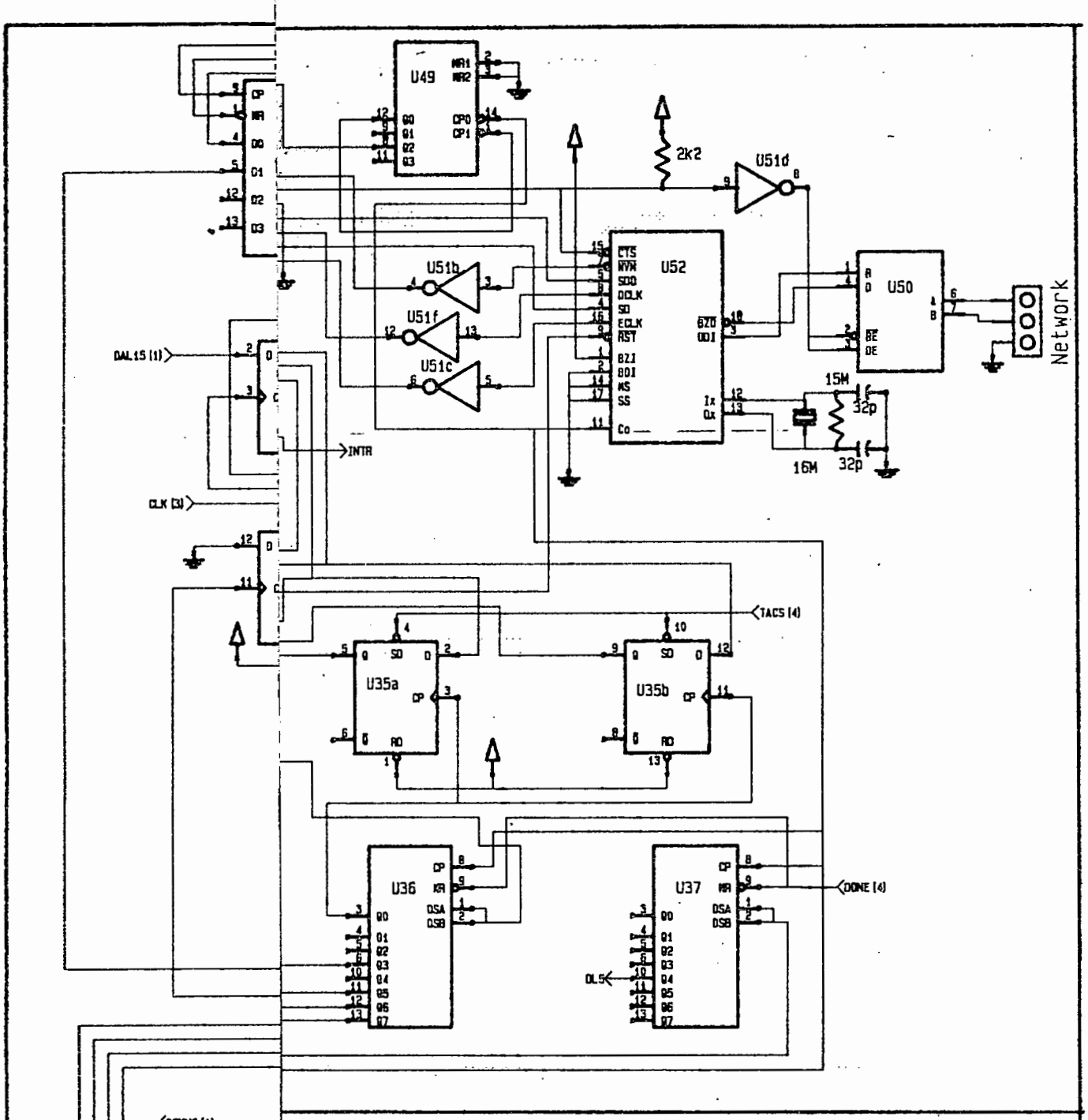
PDP 11/23 LAN Card

Sheet 2 of 5	Designed: QPM
September 85	Drawn: QPM



PDP 11/23 LAN Card

Sheet 3 of 5	Designed: QPM
September 85	Drawn: QPM



PDP 11/23 LAN Card

Sheet 4 of 5	Designed: QPM
September 85	Drawn: QPM

GND	
10	
10	
10	
10	
10	
10	
7	
8	
8	
8	
7	
8	
12	
12	
12	
10	
12	
10	
8	
7	
10	
5	
7	
10	

PDP11/23 LAN Card

Sheet 5 of 5	Designed: QPM
September 85	Drawn: QPM

The PAL equations for the PDP LAN card are given below. In PAL 4 the := means that it is a clocked output, clocked on the input specified in brackets. As before a / means negative logic.

PAL 1

```
IN1 = BDIN * INTROE * WDDAT1  +
        BDIN * INTROE * CNTCLK1  +
          WDDAT1 * CNTCLK1 * INTROE

IN2 = BDIN * INTROE * WDDAT1  +
        BDIN * INTROE * CNTCLK1  +
          WDDAT1 * CNTCLK1 * INTROE

A = /BDOUT * /CNTCS1

RE = /BDIN * /WDDAT1

WE = /BDOUT * /WDDAT1

CLK123 = /CNTCLK1 * /BDIN * READ  +
           /CNTCLK1 * /BDOUT * /READ  +
             /CNTCS1 * /BDOUT

CNTCS = CS * /DAL1 * /DAL2

CNTCLK = CS * DAL1 * /DAL2

WDADR = CS * /DAL1 * DAL2

WDDAT = CS * DAL1 * DAL2 * /TACS
```

PAL 2

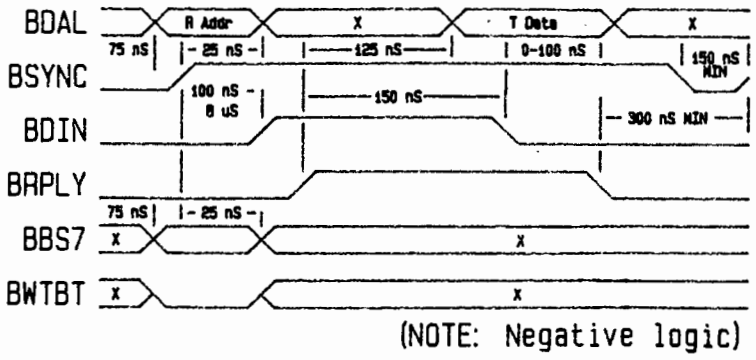
```
DLØ = /TACS  +  /T8
```

```
DMAWE = /T8 * /DRQO1

DMARE = /T8 * T4 * /DRQI1

ADREN = /IDLE * TACS * T6

CLK175 = /C16M * IDLE

READY = /HOSTOK

HOSTOK = /HOSTS  +  TACS  +  IDLE

DACK = TACS * /IDLE * /T8

IDLE = /TACS * /HOSTS

TACRQ = DRQI * DRQO
```

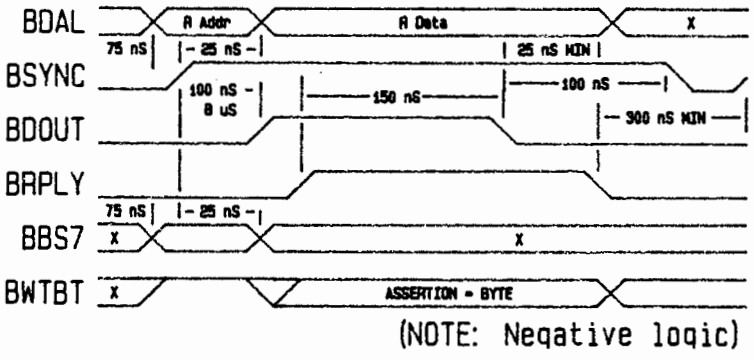
PAL 3

```
HOSTRQ = CNTCLK1 * WDDAT1

WDDOUT = /WDDAT1 * /BDIN  +  /DMAWE

SELWE = /DMAWE * /CSEN * /T7

SELRE = /DMARE * /CSEN

RAMWE = /CNTCLK1 * /CSEN * /BDOUT

RAMOE = /CNTCLK1 * /CSEN * /BDIN

RAMAE = /CNTCLK1 * /BDIN  +  /CNTCLK1 * /BDOUT

RPLCLR = WDDAT1 * CNTCLK1  +  BDIN * BDOUT
```

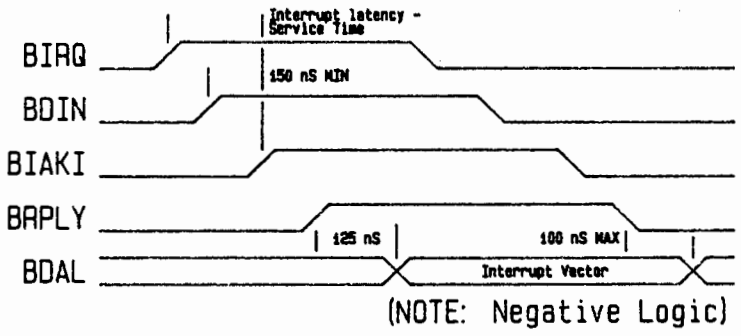
```
RPLIN = BDIN * BDOUT

BRPLY = QF * /READY  +  /INTROE  +
        /BDOUT * /CNTCS1


PAL 4

WDDOC := WDOC1  (C16M)

WDOC1 := /WDDAT1 * /BDOUT  +  /DMARE  (C16M)

WDDG = DMARE * WDDAT1 * WDDOC  +
       DMARE * BDOUT * WDDOC

REGSEL = /WDADR1 * /BDOUT

DONE = /HOSTRQ * HOSTS  +  DL5  +  /RESET


PAL 5

INTROE = /BIAKI * /BIAKI1 * INT5 * INT6 * INT7

BIAKO = /BIAKI * /INT5  +  /BIAKI * /INT6
        +  /BIAKI * /INT7

CSEN = /CSEN1  +  /DACK

DIO = /RAMOE  +  /RE
```



Q-Bus Read Timing



Q-Bus Write Timing



Q-Bus Interrupt Timing

APPENDIX H

The PDP LAN Card Test Software

```
                        .ENABL  LC
                        .TITLE PDP LAN Card Software
                ;-----------------------------------------------------------------------
                ;
                ;               LAN TEST PROGRAM
                ;
                ;-----------------------------------------------------------------------
                ;       Author          : Q.P. Mc Grath
                ;       Language        : LSI 11 Machine Code
                ;       Target System   : DEC Falcon (Running LSI 11 processor)
                ;       Date            : 19 September 1985
                ;
                ;
                ; The purpose of this program is to test the hardware constructed for the
                ; PDP 11/23. The user is given the choice of receiving the message or
                ; sending it to a specified node. The message is then displayed on the
                ; console.  Messages received from the network are automatically displayed
                ; on the console.
                ;
                ; The registers are as follows:
                ;
                ;       1. RAMADDR    - This is the address counter for a ram read or
                ;                       write.  The top bit gives the type of function
                ;                       that will follow.  If A15 is set a Write operation
                ;                       will be done, if A15 is reset then a Read follows.
                ;
                ;       2. RAMDATA    - Addressing this byte enables either a Read from
                ;                       or Write to the RAM.  The function chosen is
                ;                       determined by the status of A15.
                ;
```

```
32                              ;       3. WDREGADDR - Addressing this address sets up the register address
33                              ;                      on the WD2840.  These registers are not addressed
34                              ;                      directly to save IO address space.
35                              ;
36                              ;       4. WDREGDATA - This is the address used to read or write to or
37                              ;                      from the WD2840 registers.
38                              ;
39                              ; The Interrupt on the WD2840 causes an interrupt to address 200 Octal.
40                              ;
41                              ; The Line Time clock (LTC) causes an interrupt to 100 Octal.
42                              ;
43                              ;-------------------------------------------------------------------------
```

```
                                ;
                                ;-------------------------------------------------------------------------------
                                ;
                                ;                       Equates
                                ;
                                ;-------------------------------------------------------------------------------
000001                          TRUE    =       1
000000                          FALSE   =       0

                                ;-------------------------------------------------------------------------------
                                ;
                                ; Ascii equates

000000                          NUL     =       0
000032                          EOT     =       32
000033                          ESC     =       33
000005                          ENQ     =       5
000032                          ETX     =       32
000010                          BS      =       10
000040                          SPC     =       40
000032                          CNTRLZ  =       32
000015                          CR      =       15
000012                          LF      =       12
000177                          DEL     =       177

```

```
                                ;-------------------------------------------------------------------------------
                                ;
                                ; Console Equates

                        177560  RCSR1   =       177560
                        177562  RBUF1   =       177562
                        177564  XCSR1   =       177564
                        177566  XBUF1   =       177566
                                ;
                        000001  RECCHAR =       1

```

```
                        ;-----------------------------------------------------------------------
                        ;
                        ; WD2840 Equates
                        
000001                  SA      =       001             ; Source address for the WD board
000500                  FRAMELEN =      320.            ; The frame length for the WD
000004                  BSIZE   =       <FRAMELEN/64.>-1 ; Buffer size
001600                  RXBASE  =       400+<2*FRAMELEN> ; Base of receiver buffers
000036                  TIMERND =       30.             ; The network dead timer length (250 mS)
000020                  ACKRESP =       16.             ; Response time for ACK (500 uS)
000000                  CBLOCK  =       0               ; Position of the Control block
000001                  MYID    =       SA              ; My Address
000000                  CYSKIP  =       0               ; Number of cycles skipped at start up
000005                  MAXFRAMES =     5               ; Max number of frames transmitted per token
000000                  TXFCB   =       B 00000000      ; Frame control byte
000200                  WTXFCB  =       B 10000000      ; Frame control byte - wait for ACK
000040                  SCANFCB =       B 00100000      ; Scan FCB - transparent
                        ;
177140                  BASE      =     177140          ;
177140                  RAMADDR   =     BASE + 00       ; The hardware counter
177142                  RAMDATA   =     BASE + 02       ; Data register
177144                  WDREGADDR =     BASE + 04       ; The address latch
177146                  WDREGDATA =     BASE + 06       ; WD2840 Status Registers
```

```
                        ;    WD2840 Status Register Equates
                        ;
000000                  CR0     =       0               ; Control Register 0
000001                  CR1     =       1               ; Control Register 1
000002                  SR0     =       2               ; Status Register 0
000003                  IR0     =       3               ; Interrupt Register
000004                  SR1     =       4               ; Status Register 1
000005                  SR2     =       5               ; Status Register 2
000006                  CTR0    =       6               ; Counter Register 0
000007                  NA      =       7               ; Next Address Register
000010                  TA      =       10              ; ACK timer Register
000011                  TD      =       11              ; Network Dead Timer
000012                  CBPH    =       12              ; Control Block Pointer (MSB)
000013                  CBPL    =       13              ; Control Block Pointer (LSB)
000014                  NAR     =       14              ; Next Address Request Register
000015                  AHOLT   =       15              ; Initial Hold-off
000016                  TXLT    =       16              ; Max Number of frames per Token
000017                  MA      =       17              ; My Address Register
                        ;
                        ; Bits in the Interrupt Register (IR0)
                        ;
000001                  ITD     =       B 00000001      ; Network dead bit
000002                  ITA     =       B 00000010      ; Acknowledge timeout
000004                  ITOK    =       B 00000100      ; Token received
000010                  IREC    =       B 00001000      ; Data frame received
000020                  ITRAN   =       B 00010000      ; Data frame transmitted
000040                  INS     =       B 00100000      ; New successor found
000100                  IROR    =       B 01000000      ; Receiver over run error
000200                  ITERR   =       B 10000000      ; Transmitter error
```

```
                        ;-----------------------------------------------------------------------
                        ;
                        ;               IEEE Equate
                        ;
000003                  UI      =       B 00000011      ; Unnumbered Information frame
```

```
                                ;*****************************************************************
                                ;
                                ;                       Set the Origin
                                ;
                                ;*****************************************************************
                                ;
000000                                  .PSECT
        000500'                         .=.+500                 ; Set the origin at 500 octal
                                ;
                                ;=================================================================
                                ;
                                ;
                                ;               Main program
                                ;
                                ;*****************************************************************
000500                          MAIN:
000500  012706  006650'                  MOV     #STACK,SP       ; Set up the stack pointer
000504  004767  001072                  JSR     PC,INITNET      ; Initialise the network
000510                          ML10:                           ;
000510  012701  005406'                  MOV     #MSG7,R1        ; Ask the user what to do
000514  004767  001034                  JSR     PC,MSGOUT       ;
000520                          ML12:                           ;
000520  133727  177560  000200          BITB    @#RCSR1,#B 10000000 ; Is there a character from the console?
000526  001036                          BNE     ML14            ; Yes - test to see the users requirements
                                                                ;
000530  136727  005571  000010          BITB    IRDATA,#IREC    ; If the RXDATAFRAME flag is
000536  001027                          BNE     ML13            ;
000540  136727  005561  000100          BITB    IRDATA,#IROR    ; A receive error?
000546  001023                          BNE     ML13            ;
```

```
173                                                     ;
174 000550  016700  005554                  MOV     READLAST,R0     ;
175 000554  062700  000500                  ADD     #FRAMELEN,R0    ; Point at the new buffer
176 000560  026700  005546                  CMP     TOPADDR,R0      ; Is the pointer at the top of memory?
177 000564  100002                          BPL     ML121           ; No - then continue
178 000566  012700  001600                  MOV     #RXBASE,R0      ; Else - set R0 to the base of memory
179 000572                          ML121:                          ;
180 000572  052700  100000                  BIS     #100000,R0      ; Indicate that a read operation follows
181 000576  005200                          INC     R0              ;
182 000600  010037  177140                  MOV     R0,@#RAMADDR    ;
183 000604  113701  177142                  MOVB    @#RAMDATA,R1    ; Read the data at FSB
184 000610  130127  000200                  BITB    R1,#B 10000000 ; If zero then return
185 000614  001741                          BEQ     ML12            ;
186 000616                          ML13:                           ;
187 000616  004767  000074                  JSR     PC,RECMSG       ;     set then get the frame(s)
188 000622  000732                          BR      ML10            ;
189 000624                          ML14:                           ;
190 000624  112767  000001  005502          MOVB    #1,RBUF         ; Wait for a single letter response
191 000632  112704  000015                  MOVB    #CR,R4          ; Terminating character
192 000636  004767  000256                  JSR     PC,CIN          ;
193 000642  116701  006002                  MOVB    RBUFI,R1        ; Read the response
194 000646  142701  000200                  BICB    #B 10000000,R1 ; Clear the top bit
195 000652  122701  000124                  CMPB    #'T,R1          ; T - output
196 000656  001403                          BEQ     ML15            ;
197 000660  122701  000164                  CMPB    #'t,R1          ;
198 000664  001003                          BNE     ML20            ;
199 000666  004767  000132          ML15:   JSR     PC,SNDMSG       ;
200 000672  000706                          BR      ML10            ;
```

```
 202 000674                          ML20:
 203 000674  122701  000122                  CMPB    #'R,R1          ; R - input
 204 000700  001403                          BEQ     ML30            ;
 205 000702  122701  000162                  CMPB    #'r,R1          ;
 206 000706  001300                          BNE     ML10            ;
 207 000710                          ML30:
 208 000710  004767  000002                  JSR     PC,RECMSG       ;
 209 000714  000675                          BR      ML10            ;
```

```
                        ;------------------------------------------------------------------------
                        ;
                        ;                   Message from the network
                        ;
                        ;------------------------------------------------------------------------
                        ; NAME         : RECMSG
                        ; INPUTS       : Message from the network
                        ; OUTPUTS      : Memory INBUF - containing the message received
                        ; CALLS        : MSGOUT, IICSA
                        ; DESTROYS     : R1, R2, Flags
                        ; DESCRIPTION : Reads the message in from the network adjusting the
                        ;               Source address into ASCII and then prints the message.
                        ;
                        ;------------------------------------------------------------------------
000716                  RECMSG:
000716  012701  005514'          MOV     #MSG8,R1        ; Enable the user to leave
000722  004767  000626           JSR     PC,MSGOUT       ;
000726                  RML10:
000726  133727  177560  000200   BITB    @#RCSR1,#B 10000000 ; See if there is a character
000734  001032                   BNE     RMEXIT          ;
000736                  RML25:                           ;
000736  142767  000010  005361   BICB    #IREC,IRDATA    ; Clear the flags
000744  142767  000100  005353   BICB    #IROR,IRDATA    ;

                        ; Now go and receive the message

000752                  RML30:
000752  004767  002346           JSR     PC,READWDBUF    ;
000756  122767  000001  005331   CMPB    #TRUE,NODATA    ; Was data read?
000764  001416                   BEQ     RMEXIT          ;  No - then exit
000766  004767  000464           JSR     PC,IICSA        ; Reverse the ASCII procedure
```

```
243 000772  012701  005542'                 MOV     #MSG9,R1          ; Write out the source code
244 000776  004767  000552                  JSR     PC,MSGOUT         ;
245 001002  012701  005615'                 MOV     #INBUF+1,R1       ; Write out the message received
246 001006  004767  000542                  JSR     PC,MSGOUT         ;
247 001012  122767  000001  005274          CMPB    #TRUE,NOMOREDATA  ; Any more data?
248 001020  001354                          BNE     RML30             ; Loop if there is more data
249 001022                          RMEXIT:
250 001022  000207                          RTS     PC                ;
```

```
                        ;-----------------------------------------------------------------------
                        ;
                        ;                   Send the message over the network
                        ;
                        ;-----------------------------------------------------------------------
                        ; NAME         : SNDMSG
                        ; INPUTS       : None
                        ; OUTPUTS      : Message over the network
                        ; CALLS        : MSGOUT, CIN, UNASCII, FILTXBUF
                        ; DESTROYS     : R1, Flags
                        ; DESCRIPTION  : Prompts the user for the destination node, and then the
                        ;                user is asked to input the message to be transmitted.
                        ;                This procedure will then instruct FILTXBUF to write the
                        ;                message received to the next buffer for transmission.
                        ;
                        ;-----------------------------------------------------------------------
001024                  SNDMSG:
001024  012701  004756'          MOV     #MSG5,R1        ; Prompt the user for details
001030  004767  000520            JSR     PC,MSGOUT       ;
001034  112767  000003  005272    MOVB    #3,RBUF         ; Set up the max characters expected
001042  112704  000015            MOVB    #CR,R4          ; Terminating character
001046  004767  000046            JSR     PC,CIN          ; Wait for the response from the user
001052  004767  000220            JSR     PC,UNASCII      ; Convert the ascii destination to a
                                                          ;   hex character
001056  012701  005203'           MOV     #MSG6,R1        ; Ask for the message now
001062  004767  000466            JSR     PC,MSGOUT       ;
001066  012767  001212  005240    MOV     #650.,RBUF      ; Accept a frame of max length 650 bytes
001074  112704  000032            MOVB    #CNTRLZ,R4      ; Terminating character
001100  004767  000014            JSR     PC,CIN          ;
001104  112767  000003  005206    MOVB    #UI,CONTTYPE    ; Type of IEEE control byte
001112  004767  001526            JSR     PC,FILTXBUF     ; Fill in the network buffer
001116  000207                    RTS     PC              ;
```

```
                        ;-------------------------------------------------------------------------
                        ;
                        ;                       Console Input routine
                        ;
                        ;-------------------------------------------------------------------------
                        ; NAME         : CIN
                        ; INPUTS       : R4 - Termination character
                        ;                Memory MAXNO - containing the max number of characters
                        ;                               allowed.
                        ; OUTPUTS      : Memory NOENT - containing the number of characters read
                        ;                        BUF   - containing the characters read in from
                        ;                               the console.
                        ; CALLS        : MSGOUT
                        ; DESTROYS     : Flags
                        ; DESCRIPTION  : Reads in the characters from the console comparing each
                        ;                with CR and DEL. A CR will terminate the input, after
                        ;                which the number of characters read in from the console
                        ;                is entered into the NOENT field. A DEL character will
                        ;                cause the message <BS>,<SP>,<BS> to be sent out. If the
                        ;                number of characters input is equal to MAXNO the input
                        ;                is terminated.
                        ;
                        ;-------------------------------------------------------------------------
001120                  CIN:
001120  010246                  MOV     R2,-(SP)        ; Save all the registers which are to
001122  010146                  MOV     R1,-(SP)        ;    used
001124  010046                  MOV     R0,-(SP)        ;

001126  012701  006650'         MOV     #RBUFI,R1       ; Set up the pointer to the input buffer
001132  005002                  CLR     R2              ; Zero the counter
```

```
                                        ; To deal with the special case of the DEL
                                        ;
001222                                  CIDEL:
001222  010146                                  MOV     R1,-(SP)        ; Save current status
001224  012701  005607'                         MOV     #DELMSG,R1      ;
001230  004767  000320                          JSR     PC,MSGOUT       ; Send BS,SP,BS to console
001234  012601                                  MOV     (SP)+,R1        ; Restore the data pointer
001236  005302                                  DEC     R2              ; Adjust the counter
001240  000735                                  BR      CIL05           ; Wait for a new character
                                        ;
                                        ; Point of exit
                                        ;
001242                                  CIEXIT:
001242  010267  005070                          MOV     R2,RBUFD        ; Write the counter to memory
001246  112711  000000                          MOVB    #NUL,(R1)       ; Sign off the message in the buffer
001252  012600                                  MOV     (SP)+,R0        ; Restore the registers used to
001254  012601                                  MOV     (SP)+,R1        ;    their original value
001256  012602                                  MOV     (SP)+,R2        ;
001260  000207                                  RTS     PC              ;
```

```
                        ;-------------------------------------------------------------
                        ;
                        ;                Delay routine
                        ;
                        ;-------------------------------------------------------------
                        ; NAME         : DELAY
                        ; INPUTS       : None
                        ; OUTPUTS      : None
                        ; CALLS        : None
                        ; DESTROYS     : Flags
                        ; DESCRIPTION  : Delays the program for a short time.
                        ;
                        ;-------------------------------------------------------------
001262                  DELAY:
001262  010046                  MOV     R0,-(SP)        ;
001264  012700  023420          MOV     #10000.,R0      ; Counter
001270                  DL10:
001270  077001                  SOB     R0,DL10         ;
001272  012600                  MOV     (SP)+,R0        ; Restore register
001274  000207                  RTS     PC              ;
```

```
                        ;------------------------------------------------------------------------
                        ;
                        ;                   Untangle the ascii characters
                        ;
                        ;------------------------------------------------------------------------
                        ; NAME         : UNASCII
                        ; INPUTS       : Memory - RBUF containing up to two ASCII characters
                        ; OUTPUTS      : Memory - DESTADDR containing the destination address
                        ; CALLS        : DOASCIICON
                        ; DESTROYS     : Flags
                        ; DESCRIPTION  : Converts the ASCII digits read in via CIN into a
                        ;                destination address.
                        ;
                        ;------------------------------------------------------------------------
001276                  UNASCII:
001276  010246                MOV     R2,-(SP)        ; Save all the registers which are to
001300  010146                MOV     R1,-(SP)        ;    used
001302                  UA10:
001302  016701  005030        MOV     RBUFD,R1        ; Read in the number of characters read
001306  022701  000002        CMP     #2,R1           ;
001312  001014                BNE     UA30            ;

                        ; If two digits

001314  116701  005330        MOVB    RBUFI,R1        ;
001320  004767  000112        JSR     PC,DOASCIICON   ; Do the ascii conversion
001324                  UA20:
001324  006301                ASL     R1              ; *16
001326  006301                ASL     R1              ;
001330  006301                ASL     R1              ;
001332  006301                ASL     R1              ;
```

```
 420 001334  110102                        MOVB    R1,R2           ;
 421 001336  116701  005307                MOVB    RBUFI+1,R1      ; Read in the next character
 422 001342  000404                        BR      UA40            ;
 423
 424                                 ; If only a single digit was read in
 425
 426 001344                          UA30:
 427 001344  112702  000000                MOVB    #NUL,R2         ; Set up the first digit as zero
 428 001350  116701  005274                MOVB    RBUFI,R1        ;
 429
 430                                 ; Now both possibilities converge
 431
 432 001354                          UA40:
 433 001354  004767  000056                JSR     PC,DOASCIICON   ;
 434 001360  142701  000360                BICB    #B 11110000,R1 ; Clear the top 4 bits
 435 001364  074102                        XOR     R1,R2           ; Combine the two nibbles
 436 001366  122702  000000                CMPB    #0.,R2          ;
 437 001372  001405                        BEQ     UAERROR         ; If 0 then error.
 438 001374  110267  004721                MOVB    R2,DESTADDR     ; Save result
 439
 440                                 ; All done
 441
 442 001400                          UAEXIT:
 443 001400  012601                        MOV     (SP)+,R1        ; Restore the registers used to
 444 001402  012602                        MOV     (SP)+,R2        ;    their original value
 445 001404  000207                        RTS     PC              ;
```

```
                                ; Error in destination address
                                ;
001406                          UAERROR:
001406  012701  005276'                 MOV     #MSG65,R1       ; Tell the user of his mistake
001412  004767  000136                  JSR     PC,MSGOUT       ;
                                ;
001416  112767  000003  004710          MOVB    #3,RBUF         ; Read in the new address
001424  112704  000015                  MOVB    #CR,R4          ; Terminating character
001430  004767  177464                  JSR     PC,CIN          ;
                                ;
001434  000722                          BR      UA10            ; Try again
                                ;
                                ;-------------------------------------------------------------------
001436                          DOASCIICON:
001436  130127  000100                  BITB    R1,#B 01000000 ; Is bit six set (i.e. A-F)
001442  001402                          BEQ     DACEXIT         ;
                                ;
001444  062701  000011                  ADD     #9.,R1          ; Adjust A-F
                                ;
001450                          DACEXIT:
001450  142701  000360                  BICB    #B 11110000,R1 ; Clear the top nibble and return
001454  000207                          RTS     PC              ;
```

```
470                             ;-----------------------------------------------------------------------
471                             ;
472                             ;                   Conversion routine for the SA
473                             ;
474                             ;-----------------------------------------------------------------------
475                             ; NAME        : IICSA
476                             ; INPUTS      : Memory INBUF - containing Source address in HEX
477                             ; OUTPUTS     : Memory MSG9I - containing ASCII Source Address
478                             ; CALLS       : None
479                             ; DESTROYS    : R1, Flags
480                             ; DESCRIPTION : Simply converts the SA from HEX to ASCII code.
481                             ;
482                             ;-----------------------------------------------------------------------
483 001456                     IICSA:
484 001456  116701  004132             MOVB    INBUF,R1          ; Read SA
485 001462  110102                     MOVB    R1,R2             ;
486 001464  006201                     ASR     R1                ; MSB first
487 001466  006201                     ASR     R1                ;
488 001470  006201                     ASR     R1                ;
489 001472  006201                     ASR     R1                ;
490 001474  142701  000360             BICB    #B 11110000,R1 ; Clear the top nibble
491 001500  120127  000012             CMPB    R1,#B 00001010 ;
492 001504  100402                     BMI     IIL10             ;
493 001506  062701  000007             ADD     #7.,R1            ;
494 001512                     IIL10:
495 001512  062701  000060             ADD     #B 00110000,R1 ; For an ascii digit
496 001516  110167  004055             MOVB    R1,MSG9I          ; Insert value in msg
497
498 001522  110201                     MOVB    R2,R1             ;
499 001524  142701  000360             BICB    #B 11110000,R1 ;
500 001530  120127  000012             CMPB    R1,#B 00001010 ;
```

```
    502 001534  100402                        BMI     IIL20           ;
    503 001536  062701  000007                ADD     #7.,R1          ;
    504 001542                          IIL20:
    505 001542  062701  000060                ADD     #B 00110000,R1 ;
    506 001546  110167  004026                MOVB    R1,MSG9I+1      ;
    507
    508 001552  000207                        RTS     PC              ;
```

```
                                ;*******************************************************************************
                                ;
                                ;                       Console Message out routine
                                ;
                                ;*******************************************************************************
                                ; NAME         : MSGOUT
                                ; INPUTS       : R1 - pointing at message
                                ; OUTPUTS      : Message to the console
                                ; CALLS        : None
                                ; DESTROYS     : R1, Flags
                                ; DESCRIPTION  : Writes the message to the console, checking for a NUL
                                ;                which indicates the end of message.
                                ;
                                ;*******************************************************************************
001554                          MSGOUT:
001554  133727  177564  000200          BITB    @#XCSR1,#B 10000000     ;
001562  001774                        BEQ     MSGOUT                  ;
001564  121127  000000                CMPB    (R1),#NUL               ; End of message?
001570  001403                        BEQ     MOEXIT                  ;
001572  112137  177566                MOVB    (R1)+,@#XBUF1           ; Transmit next char
001576  000766                        BR      MSGOUT                  ;
                                                                        ;
001600                          MOEXIT:                                 ;
001600  000207                        RTS     PC                      ;
```

```
                        ;=============================================================================
                        ;
                        ;                       MAC Layer Routines
                        ;
                        ;=============================================================================
                        ;-----------------------------------------------------------------------------
                        ;
                        ;                       1. WD2840 Initialisation
                        ;
                        ;-----------------------------------------------------------------------------
                        ; NAME          : INITNET
                        ; INPUTS        : None
                        ; OUTPUTS       : None
                        ; CALLS         : MSGOUT
                        ; DESTROYS      : R1, R2, Flags
                        ; DESCRIPTION   : INITNET sets up the parameters as required by the WD 2840
                        ;                 and informs the user of the progress. If the network is
                        ;                 dead then the LAN controller will seek to start it up. If
                        ;                 the network is alive then the Controller will check to
                        ;                 see if there is a duplicate address if so it will inform
                        ;                 the user and stop. Otherwise it will enter the ring on
                        ;                 the next token pass.
                        ;
                        ;-----------------------------------------------------------------------------
001602                  INITNET:

                        ; Welcome the user

001602  012701  004414'          MOV     #MSG1,R1                ; Write the message to the
001606  004767  177742          JSR     PC,MSGOUT               ;   terminal.
```

```
                                        ; Set up the control block
                                        
001612  012737  077777  177140          MOV     #077777,@#RAMADDR       ; Write to address 0
001620  012701  001600                  MOV     #RXBASE,R1              ;
001624  000301                          SWAB    R1                      ; Write out MSB of address first
001626  110137  177142                  MOVB    R1,@#RAMDATA            ;
001632  000301                          SWAB    R1                      ; Now LSB
001634  110137  177142                  MOVB    R1,@#RAMDATA            ;
                                        
001640  012701  000400                  MOV     #000400,R1              ; Then the transmit address
001644  000301                          SWAB    R1                      ; The MSB <> 0 thus 400
001646  110137  177142                  MOVB    R1,@#RAMDATA            ;
001652  000301                          SWAB    R1                      ;
001654  110137  177142                  MOVB    R1,@#RAMDATA            ;
001660  112737  000004  177142          MOVB    #BSIZE,@#RAMDATA        ; Set up the buffer size
                                                                        ;
001666  012701  000013                  MOV     #11.,R1                 ; Zero the following 11 places
001672  112737  000000  177142  IL01:   MOVB    #NUL,@#RAMDATA          ;  for event counters
001700  005301                          DEC     R1                      ;
001702  001373                          BNE     IL01                    ;
                                        
                                        ; Set up the links for the receive buffers
                                        
001704  012700  001577                  MOV     #RXBASE-1,R0            ; Find the initial buffer
001710  010001                          MOV     R0,R1                   ;
001712  005201                          INC     R1                      ;
001714                          IL10:
001714  062701  000500                  ADD     #FRAMELEN,R1            ; Get the next address
001720  030127  020000                  BIT     R1,#020000              ;
001724  001020                          BNE     IL20
001726  042700  100000                  BIC     #100000,R0              ; Indicate that a write operation
```

```
598 001732  010037  177140              MOV     R0,@#RAMADDR            ;    follows
599 001736  000301                      SWAB    R1                      ; Write the MSB first
600 001740  110137  177142              MOVB    R1,@#RAMDATA            ;
601 001744  000301                      SWAB    R1                      ;
602 001746  110137  177142              MOVB    R1,@#RAMDATA            ;
603 001752  112737  000000  177142      MOVB    #0,@#RAMDATA            ; Zero the FSB
604 001760  062700  000500              ADD     #FRAMELEN,R0            ; Adjust the pointer
605 001764  000753                      BR      IL10
606 001766                          IL20:
607 001766  162700  000500              SUB     #FRAMELEN,R0            ; Get previous buffer
608 001772  005200                      INC     R0                      ;
609 001774  010067  004332              MOV     R0,TOPADDR              ; Save this for later
610 002000  005300                      DEC     R0                      ;
611 002002  010037  177140              MOV     R0,@#RAMADDR            ; Zero the last link
612 002006  112737  000000  177142      MOVB    #0,@#RAMDATA            ;
613 002014  112737  000000  177142      MOVB    #0,@#RAMDATA            ;
614 002022  112737  000000  177142      MOVB    #0,@#RAMDATA            ; And the FSB
615
616                                 ; Initialise registers and flags
617
618 002030  012767  001100  004272      MOV     #RXBASE-FRAMELEN,READLAST ; Set up the input buffer pointer
619 002036  112767  000002  004252      MOVB    #2,TXBUF                ; Second buffer used
620
621                                 ; Set up the registers
622
623 002044  012705  000010              MOV     #TA,R5                  ; Save the present WD reg addressed
624 002050  110537  177144              MOVB    R5,@#WDREGADDR          ;
625 002054  112737  000020  177146      MOVB    #ACKRESP,@#WDREGDATA    ; Set up ACK timer
626 002062  005205                      INC     R5                      ; R5 points at TD register
627 002064  110537  177144              MOVB    R5,@#WDREGADDR          ;
```

```
629 002070  112737  000036  177146          MOVB    #TIMERND,@#WDREGDATA    ; Set up network dead timer
630 002076  012701  000000                  MOV     #CBLOCK,R1              ;
631 002102  012705  000013                  MOV     #CBPL,R5                ;
632 002106  110537  177144                  MOVB    R5,@#WDREGADDR          ; Set up the address of register
633 002112  110137  177146                  MOVB    R1,@#WDREGDATA          ; Write out the control
634 002116  000301                          SWAB    R1                      ;    block address
635 002120  012705  000012                  MOV     #CBPH,R5                ;
636 002124  110537  177144                  MOVB    R5,@#WDREGADDR          ; Set up the address of register
637 002130  110137  177146                  MOVB    R1,@#WDREGDATA          ;
638 002134  012705  000014                  MOV     #NAR,R5                 ;
639 002140  110537  177144                  MOVB    R5,@#WDREGADDR          ;
640 002144  112737  000002  177146          MOVB    #MYID+1,@#WDREGDATA     ; Next Address set up
641 002152  005205                          INC     R5                      ; R5 points at AHOLT
642 002154  110537  177144                  MOVB    R5,@#WDREGADDR          ;
643 002160   12737  000000  177146          MOVB    #CYSKIP,@#WDREGDATA     ; Miss cycles
644 002166  005205                          INC     R5                      ; Points at TXLT
645 002170  110537  177144                  MOVB    R5,@#WDREGADDR          ;
646 002174  112737  000005  177146          MOVB    #MAXFRAMES,@#WDREGDATA  ; Set the max frames
647 002202  005205                          INC     R5                      ; Points at MA
648 002204  110537  177144                  MOVB    R5,@#WDREGADDR          ; Set up the address of register
649 002210  112737  000001  177146          MOVB    #MYID,@#WDREGDATA       ; Set up my address
650
651                                 ; Do a dummy start to see if the network is dead
652
653 002216  012705  000001                  MOV     #CR1,R5                 ;
654 002222  110537  177144                  MOVB    R5,@#WDREGADDR          ;
655 002226  112737  000000  177146          MOVB    #0.,@#WDREGDATA         ; Zero the Control regs
656 002234  012705  000000                  MOV     #CR0,R5                 ;
657 002240  110537  177144                  MOVB    R5,@#WDREGADDR          ;
658 002244  112737  000000  177146          MOVB    #0.,@#WDREGDATA         ;
```

```
660 002252  012705  000005              MOV     #SR2,R5                 ;
661 002256  110537  177144              MOVB    R5,@#WDREGADDR          ;
662 002262                          DUMMYLOOP:                          ; Wait for confirmation
663 002262  004767  176774              JSR     PC,DELAY                ; Let the WD update its regs
664 002266  133727  177146  000002      BITB    @#WDREGDATA,#B 00000010 ; Wait for SR21 to clear confirming
665 002274  001372                      BNE     DUMMYLOOP               ;    a state change
666                                     ;
667 002276  012767  000000  004016      MOV     #0.,TIMERCOUNT          ; Reset the timer counter
668 002304  112767  000000  004013      MOVB    #0,IRDATA               ; Clear the interrupt register save
669                                                                     ;   space
670 002312                          DML10:
671 002312  004767  176744              JSR     PC,DELAY                ; Let the WD update its regs
672 002316  136727  004003  000001      BITB    IRDATA,#ITD             ; Has there been a NETDEAD interrupt?
673 002324  001062                      BNE     NETDEAD                 ;
674 002326  026727  003770  000017      CMP     TIMERCOUNT,#15.         ; Wait for 300 msec
675 002334  101366                      BHI     DML10                   ;
676
677                                 ; The program reaches here if the network is NOT dead
678                                 ;  so enter into the swing of things by claiming the next token
679
680 002336  012705  000001              MOV     #CR1,R5                 ;
681 002342  110537  177144              MOVB    R5,@#WDREGADDR          ;
682 002346  112737  000041  177146      MOVB    #B 00100001,@#WDREGDATA ; Enter scan mode, take new NA
683 002354  012705  000000              MOV     #CR0,R5                 ;
684 002360  110537  177144              MOVB    R5,@#WDREGADDR          ;
685 002364  112737  000120  177146      MOVB    #B 01010000,@#WDREGDATA ; Enable token interrupt and Control
686                                                                     ;   frame transmission
```

```
688                                     ; Watch for address duplication
689
690 002372  012701  004442'                     MOV     #MSG10,R1               ; Tell user of progress
691 002376  004767  177152                      JSR     PC,MSGOUT               ;
692
693 002402  142767  000004  003715              BICB    #ITOK,IRDATA            ; Clear the TOKEN RECEIVED bit in the
694                                                                             ;   interrupt strage space
695 002410  012767  000000  003704              MOV     #0.,TIMERCOUNT          ; Reset the timer counter
696 002416                              ADL10:
697 002416  012701  000002                      MOV     #2.,R1                  ; Token counter
698 002422  136727  003677  000004              BITB    IRDATA,#ITOK            ; Was there a token received?
699 002430  001002                              BNE     ADL20                   ;
700 002432  005301                              DEC     R1                      ; If token then decrement R1
701 002434  001411                              BEQ     ADL15                   ;
702 002436                              ADL20:
703 002436  026727  003660  000062              CMP     TIMERCOUNT,#50.         ; Wait for 1 sec
704 002444  101364                              BHI     ADL10                   ;
705
706 002446  012701  004577'                     MOV     #MSG2,R1                ; Tell user of progress
707 002452  004767  177076                      JSR     PC,MSGOUT               ;
708
709 002456  000411                              BR      NDL05                   ; Now get into the loop
710
711 002460                              ADL15:
712 002460  012701  004650'                     MOV     #MSG3,R1                ;
713 002464  004767  177064                      JSR     PC,MSGOUT               ;
714 002470  000000                              HALT                            ; Die as nothing else can be done
```

```
716                                     ; Get into the logical loop
717
718 002472                              NETDEAD:
719 002472  012701  004506'                     MOV     #MSG11,R1               ; Tell user of progress
720 002476  004767  177052                      JSR     PC,MSGOUT               ;
721 002502                              NDL05:
722 002502  012705  000001                      MOV     #CR1,R5                 ;
723 002506  110537  177144                      MOVB    R5,@#WDREGADDR          ;
724 002512  112737  000051  177146              MOVB    #B 00101001,@#WDREGDATA ; Enables the TAC to enter the loop
725 002520  012705  000000                      MOV     #CR0,R5                 ;
726 002524  110537  177144                      MOVB    R5,@#WDREGADDR          ;
727 002530  112737  000120  177146              MOVB    #B 01010000,@#WDREGDATA ; Enable token interrupt and control
728                                                                             ;   information frames
729
730 002536  012705  000005                      MOV     #SR2,R5                 ;
731 002542  110537  177144                      MOVB    R5,@#WDREGADDR          ;
732 002546                              NDL10:                                  ; Wait for the TAC to confirm change
733 002546  004767  176510                      JSR     PC,DELAY                ; Let the TAC update
734 002552  113701  177146                      MOVB    @#WDREGDATA,R1          ; Read the status from SR2
735 002556  142701  000374                      BICB    #B 11111100,R1          ; Only concerned with lowest two bits
736 002562  120127  000001                      CMPB    R1,#B 00000001          ; Is STATE clear, and INRING set?
737 002566  001367                              BNE     NDL10                   ; Loop until ready
738
739                                     ; Enable the transmission of frames
740
741 002570  012705  000000                      MOV     #CR0,R5                 ;
742 002574  110537  177144                      MOVB    R5,@#WDREGADDR          ;
743 002600  112737  000140  177146              MOVB    #B 01100000,@#WDREGDATA ; Disable token receive interrupts
744 002606  012705  000001                      MOV     #CR1,R5                 ;
745 002612  110537  177144                      MOVB    R5,@#WDREGADDR          ;
```

```
747 002616  113701  177146          MOVB    @#WDREGDATA,R1          ; Read from CR1
748 002622  152701  000100          BISB    #B 01000000,R1         ;
749 002626  110137  177146          MOVB    R1,@#WDREGDATA          ; Read the status
750                                                                 ;
751 002632  012701  004722'         MOV     #MSG4,R1                ;
752 002636  004767  176712          JSR     PC,MSGOUT               ;
753                                                                 ;
754 002642  000207                  RTS     PC                      ; Ready for transmission
```

```
                                ;*****************************************************************
                                ;
                                ;               2. Fill in the Network transmit buffer
                                ;
                                ;*****************************************************************
                                ; NAME         : FILTXBUF
                                ; INPUTS       : Memory DESTADDR - containing the destination address
                                ;                       RBUFI    - containing the message to be sent
                                ; OUTPUTS      : Message to the network
                                ; CALLS        : None
                                ; DESTROYS     : R0, R1, R2,Flags
                                ; DESCRIPTION : Finds the next available buffer and then fills in the
                                ;               data from the CIN buffer.
                                ;
                                ;*****************************************************************
002644                          FILTXBUF:
002644  012704  006650'                MOV     #RBUFI,R4       ; Set up the pointer to the data

                                ; Find out which buffer is to be used

002650                          FTXL0:
002650  122767  000001  003440        CMPB    #1,TXBUF        ; First or second buffer used
002656  001406                        BEQ     FTXL01          ;
002660  012701  000400                MOV     #400,R1         ; - Used second last time, so first this time
002664  112767  000001  003424        MOVB    #1,TXBUF        ;
002672  000405                        BR      FTXL05          ;
002674                          FTXL01:                         ;
002674  012701  001100                MOV     #400+FRAMELEN,R1 ; - Use second this time
002700  112767  000002  003410        MOVB    #2,TXBUF        ;
```

```
786 002706                                FTXL05:                         ;
787 002706  005301                              DEC     R1              ; For the hardware
788 002710  042701  100000                      BIC     #100000,R1      ; Write operation
789 002714  010137  177140                      MOV     R1,@#RAMADDR    ;
790 002720  112737  000000  177142              MOVB    #0,@#RAMDATA    ; Clear links and FSB
791 002726  112737  000000  177142              MOVB    #0,@#RAMDATA    ;
792 002734  112737  000000  177142              MOVB    #0,@#RAMDATA    ;
793 002742  122767  000377  003351              CMPB    #377,DESTADDR   ; Is it a broadcast frame?
794 002750  001404                              BEQ     FTXL10          ;
795 002752  112737  000200  177142              MOVB    #WTXFCB,@#RAMDATA ; Set up the control byte (Wait for ACK)
796 002760  000403                              BR      FTXL20          ;
797 002762                                FTXL10:                         ;
798 002762  112737  000000  177142              MOVB    #TXFCB,@#RAMDATA ; Dont wait for an ACK
799 002770                                FTXL20:                         ;
800 002770  016702  003342                      MOV     RBUFD,R2        ; Read the length counter
801 002774  020227  000467                      CMP     R2,#FRAMELEN-9. ; Can send max no of bytes if single buffer
802 003000  002414                              BLT     FTXL25          ;
803 003002  012703  000477                      MOV     #FRAMELEN-1,R3  ; Frame counter
804 003006  162702  000467                      SUB     #FRAMELEN-9.,R2 ; Adjust the receive buffer counter
805 003012  010267  003320                      MOV     R2,RBUFD        ;
806 003016  112767  000001  003273              MOVB    #TRUE,LONG      ; It is a long frame
807 003024  012702  000467                      MOV     #FRAMELEN-9.,R2 ; Set up frame counter
808 003030  000406                              BR      FTXL30          ;
809 003032                                FTXL25:                         ; Less than max bytes to be sent
810 003032  010203                              MOV     R2,R3           ;
811 003034  062703  000011                      ADD     #9.,R3          ; Frame counter
812 003040  112767  000000  003251              MOVB    #FALSE,LONG     ;
813 003046                                FTXL30:                         ; Fill in the WD frame
814 003046  062701  000004                      ADD     #4,R1           ; To point at the length buffer
815 003052  042701  100000                      BIC     #100000,R1      ; Writting to follow
816 003056  010137  177140                      MOV     R1,@#RAMADDR    ;
```

```
818 003062  000303                          SWAB    R3                ; MSB of length first
819 003064  110337  177142                  MOVB    R3,@#RAMDATA      ;
820 003070  000303                          SWAB    R3                ; Then LSB of length
821 003072  110337  177142                  MOVB    R3,@#RAMDATA      ;
822 003076  116737  003217  177142          MOVB    DESTADDR,@#RAMDATA ; Then DA
823 003104  112737  000001  177142          MOVB    #SA,@#RAMDATA     ;       and SA
824 003112  116737  003202  177142          MOVB    CONTTYPE,@#RAMDATA ;    and the IEEE control byte
825
826                                 ; Now send out the data
827
828 003120                           FTXL40:                           ;
829 003120  112437  177142                  MOVB    (R4)+,@#RAMDATA ; Write out data
830 003124  005302                          DEC     R2                ;
831 003126  001374                          BNE     FTXL40            ; Loop until data written
832
833                                 ; Link up and wait for transmission
834
835 003130  122767  000001  003160          CMPB    #1,TXBUF          ; Which buffers was used
836 003136  001405                          BEQ     FTXL50            ;
837 003140  012701  000400                  MOV     #400,R1           ; Set up pointers for linking
838 003144  012702  001100                  MOV     #400+FRAMELEN,R2  ;
839 003150  000404                          BR      FTXL55            ;
840 003152                           FTXL50:                           ;
841 003152  012701  001100                  MOV     #400+FRAMELEN,R1  ;
842 003156  012702  000400                  MOV     #400,R2           ;
843 003162                           FTXL55:                           ;
844 003162  042701  100000                  BIC     #100000,R1        ; Link up
845 003166  010137  177140                  MOV     R1,@#RAMADDR      ;
846 003172  110237  177142                  MOVB    R2,@#RAMDATA      ; LSB byte first
847 003176  000302                          SWAB    R2                ;
```

```
 849 003200  005301                        DEC     R1              ; To point at first link byte
 850 003202  010137  177140                 MOV     R1,@#RAMADDR    ;
 851 003206  110237  177142                 MOVB    R2,@#RAMDATA    ; Then MSB
 852 003212  000302                         SWAB    R2              ; Restore R2
 853
 854                                ; Enable transmission of data
 855
 856 003214  012705  000000                 MOV     #CR0,R5         ;
 857 003220  110537  177144                 MOVB    R5,@#WDREGADDR  ;
 858 003224  112737  000340  177146         MOVB    #B 11100000,@#WDREGDATA ; Enable data transmission
 859
 860 003232  005202                         INC     R2              ; Point at FSB
 861 003234  052702  100000                 BIS     #100000,R2      ; Read operation
 862 003240                         FTXL60:                         ;
 863 003240  010237  177140                 MOV     R2,@#RAMADDR    ;
 864 003244  113701  177142                 MOVB    @#RAMDATA,R1    ; Read FSB
 865 003250  130127  000200                 BITB    R1,#B 10000000 ; DONE set?
 866 003254  001002                         BNE     FTXL65          ; Yes - then continue
 867 003256  000001                         WAIT                    ; Wait for an interrupt
 868 003260  000767                         BR      FTXL60          ; And loop
 869 003262                         FTXL65:                         ;
 870 003262  130127  000007                 BITB    R1,#B 00000111 ; If any of the lower 3 bits set = error
 871 003266  001006                         BNE     FTXEX           ;
 872 003270  122767  000000  003021         CMPB    #FALSE,LONG     ; If LONG then loop
 873 003276  001402                         BEQ     FTXEX           ;
 874 003300  000167  177344                 JMP     FTXL0           ;
```

```
876                                     ; Finish up
877
878 003304                              FTXEX:                                  ;
879 003304  012705  000000                      MOV     #CR0,R5                 ;
880 003310  110537  177144                      MOVB    R5,@#WDREGADDR          ;
881 003314  112737  000140  177146              MOVB    #B 01100000,@#WDREGDATA ; Disable transmit
882 003322  000207                              RTS     PC                      ;
```

```
                        ;-------------------------------------------------------------
                        ;
                        ;               3. Read the WD buffer
                        ;
                        ;-------------------------------------------------------------
                        ; NAME         : READWDBUF
                        ; INPUTS       : Frame in WD local memory
                        ; OUTPUTS      : Memory INBUF - containing the message
                        ; CALLS        : None
                        ; DESTROYS     : R0, R1, R2, Flags
                        ; DESCRIPTION : Reads the data in from the local memory on the WD board
                        ;               into main memory.
                        ;
                        ;-------------------------------------------------------------
003324                  READWDBUF:
003324  016700  003000          MOV     READLAST,R0     ;
003330  062700  000500          ADD     #FRAMELEN,R0    ; Point at the new buffer
003334  026700  002772          CMP     TOPADDR,R0      ; Is the pointer at the top of memory?
003340  100002                  BPL     RBL01           ; No - then continue
003342  012700  001600          MOV     #RXBASE,R0      ; Else - set R0 to the base of memory
003346                  RBL01:                          ;
003346  010004                  MOV     R0,R4           ; As a check byte

                        ; Check to see if a frame was received

003350  052700  100000          BIS     #100000,R0      ; Indicate that a read operation follows
003354  005200                  INC     R0              ;
003356  010037  177140          MOV     R0,@#RAMADDR    ;
003362  113701  177142          MOVB    @#RAMDATA,R1    ; Read the data at FSB
003366  130127  000200          BITB    R1,#B 10000000 ; If zero then return
003372  001032                  BNE     RBL05           ;
```

```
                                        ; Check the following frame to see if received correctly
                                        ;  (occurs if receiver error)
                                        ;
003374  005300                          DEC     R0              ;
003376  042700  100000                  BIC     #100000,R0      ;
003402  062700  000500                  ADD     #FRAMELEN,R0    ; Point at the new buffer
003406  026700  002720                  CMP     TOPADDR,R0      ; Is the pointer at the top of memory?
003412  100002                         BPL     RBL03           ; No - then continue
003414  012700  001600                  MOV     #RXBASE,R0      ; Else - set R0 to the base of memory
003420                          RBL03:                          ;
003420  010004                          MOV     R0,R4           ; As a check byte
003422  005200                          INC     R0              ;
003424  052700  100000                  BIS     #100000,R0      ; Indicate that a read operation follows
003430  010037  177140                  MOV     R0,@#RAMADDR    ;
003434  113701  177142                  MOVB    @#RAMDATA,R1    ; Read the data at FSB
003440  130127  000200                  BITB    R1,#B 10000000 ; If zero then return
003444  001005                          BNE     RBL05           ;
                                        ;
                                        ; Definately no data
                                        ;
003446  112767  000001  002641          MOVB    #TRUE,NODATA    ; If no frames received
003454  000167  000612                  JMP     RBE01           ;
                                        ;
                                        ; Or receive the data
                                        ;
003460                          RBL05:                          ;
003460  005300                          DEC     R0              ;
003462  042700  100000                  BIC     #100000,R0      ; Indicate that a write operation follows
003466  010067  002634                  MOV     R0,READFIRST    ; Save the first buffer read
003472  010067  002632                  MOV     R0,READLAST     ;  and the last buffer read
```

```
947 003476  005200                         INC     R0              ;
948 003500  010037  177140                  MOV     R0,@#RAMADDR    ;
949 003504  112737  000000  177142          MOVB    #NUL,@#RAMDATA  ; Clear FSB
950
951                             ; Read the length word
952
953 003512  005200                          INC     R0              ;
954 003514  005200                          INC     R0              ; Point at the length field
955 003516  052700  100000                  BIS     #100000,R0      ; Indicate that a read operation follows
956 003522  010037  177140                  MOV     R0,@#RAMADDR    ; Read the length word
957 003526  113701  177142                  MOVB    @#RAMDATA,R1    ;
958 003532  042701  177400                  BIC     #177400,R1      ; Clear top byte
959 003536  000301                          SWAB    R1              ;
960 003540  113702  177142                  MOVB    @#RAMDATA,R2    ;
961 003544  042702  177400                  BIC     #177400,R2      ; Clear the top byte
962 003550  074201                          XOR     R2,R1           ; Combine the length
963 003552  162701  000007                  SUB     #7,R1           ; Adjust for the overhead bytes
964 003556  012702  005614'                 MOV     #INBUF,R2       ; Set up the input pointer
965 003562  012703  000007                  MOV     #7,R3           ; The internal buffer pointer
966
967                             ; Ready to read in the data
968
969 003566                      RBL07:
970 003566  113712  177142                  MOVB    @#RAMDATA,(R2)  ;Dummy read to adjust the pointer
971 003572                      RBL10:
972 003572  113722  177142                  MOVB    @#RAMDATA,(R2)+ ; Read the data from the WD buffer
973 003576  022702  006314'                 CMP     #ENDBUF,R2      ; At the bottom of data area?
974 003602  001001                          BNE     RBL11           ;
975 003604  005302                          DEC     R2              ; Adjust the pointer to prevent corruption
976 003606                      RBL11:                              ;
977 003606  005301                          DEC     R1              ; End of the frame?
```

```
 979 003610  003430                    BLE     RBL20           ;
 980 003612  005203                    INC     R3              ; End of the present buffer ?
 981 003614  022703  000500            CMP     #FRAMELEN,R3    ;
 982 003620  001364                    BNE     RBL10           ; Yes - then adjust things else loop
 983
 984                            ; End of buffer sequence
 985
 986 003622  005301                    DEC     R1              ; Adjust the counter for link bytes
 987 003624  005301                    DEC     R1              ;
 988 003626  001421                    BEQ     RBL20           ;
 989 003630  062704  000500            ADD     #FRAMELEN,R4    ;
 990 003634  026704  002472            CMP     TOPADDR,R4      ; Top of buffer?
 991 003640  100005                    BPL     RBL15           ;
 992 003642  012704  001600            MOV     #RXBASE,R4      ;  Yes - then point at bottom of buffers
 993 003646  010437  177140            MOV     R4,@#RAMADDR    ; Write out the new address
 994 003652  000402                    BR      RBL17           ;
 995 003654                     RBL15:                         ;
 996 003654  113712  177142            MOVB    @#RAMDATA,(R2)  ; Dummy read to adjust RAM pointer
 997 003660                     RBL17:                         ;
 998 003660  012703  000002            MOV     #2,R3           ; Reset the buffer counter
 999 003664  010467  002440            MOV     R4,READLAST     ;
1000 003670  000736                    BR      RBL07           ; Continue reading
1001
1002                            ; End of Frame sequence - Fill in the link addresses
1003
1004 003672                     RBL20:
1005 003672  112712  000000            MOVB    #NUL,(R2)       ; Sign off the message
1006                                                           ;
1007 003676  016700  002426            MOV     READLAST,R0     ; Find the first buffer read
```

```
1009 003702  010001                        MOV     R0,R1           ;
1010 003704  026700  002422                CMP     TOPADDR,R0      ; Link to next buffer first
1011 003710  001002                        BNE     RBL201          ;
1012 003712  012701  001100                MOV     #RXBASE-FRAMELEN,R1 ;
1013 003716                        RBL201: :                       ;
1014 003716  062701  000500                ADD     #FRAMELEN,R1    ;
1015 003722  042700  100000                BIC     #100000,R0      ;
1016 003726  010037  177140                MOV     R0,@#RAMADDR    ; Write LSB to the higher addr first
1017 003732  110137  177142                MOVB    R1,@#RAMDATA    ;
1018 003736  000301                        SWAB    R1              ;
1019 003740  005300                        DEC     R0              ;
1020 003742  010037  177140                MOV     R0,@#RAMADDR    ;
1021 003746  110137  177142                MOVB    R1,@#RAMDATA    ;
1022 003752  005200                        INC     R0              ;
1023 003754                        RBL21:  :                       ;
1024 003754  010001                        MOV     R0,R1           ;
1025 003756  010002                        MOV     R0,R2           ;
1026 003760  022700  001600                CMP     #RXBASE,R0      ; Bottom of buffers?
1027 003764  001004                        BNE     RBL22           ;  No - then continue as normal
1028 003766  016700  002340                MOV     TOPADDR,R0      ;  Else - point at the top buffer
1029 003772  062700  000500                ADD     #FRAMELEN,R0    ;
1030 003776                        RBL22:  :                       ;
1031 003776  162700  000500                SUB     #FRAMELEN,R0    ; Get to the link bytes
1032 004002  042700  100000                BIC     #100000,R0      ; Link the previous buffer to the present
1033 004006  005300                        DEC     R0              ; Adjust for hardware
1034 004010  010037  177140                MOV     R0,@#RAMADDR    ;
1035 004014  112737  000000  177142        MOVB    #0,@#RAMDATA    ; Zero previous buffers link
1036 004022  112737  000000  177142        MOVB    #0,@#RAMDATA    ;
1037 004030  112737  000000  177142        MOVB    #0,@#RAMDATA    ; Clear the FSB
1038 004036  005200                        INC     R0              ; Re-adjust pointer
```

```
1040 004040                        RBL25:                          ;
1041 004040  010001                        MOV     R0,R1           ;
1042 004042  022700  001600                CMP     #RXBASE,R0      ; Limits of buffers?
1043 004046  001004                        BNE     RBL27           ;
1044 004050  016700  002256                MOV     TOPADDR,R0      ;
1045 004054  062700  000500                ADD     #FRAMELEN,R0    ; Adjust for next command
1046 004060                        RBL27:                          ;
1047 004060  162700  000500                SUB     #FRAMELEN,R0    ; Get to the link bytes
1048 004064  042700  100000                BIC     #100000,R0      ; Link the previous buffer to the present
1049 004070  010037  177140                MOV     R0,@#RAMADDR    ;
1050 004074  110137  177142                MOVB    R1,@#RAMDATA    ; Write out the LSB first
1051 004100  112737  000000  177142        MOVB    #0,@#RAMDATA    ; Clear the FSB
1052 004106  000301                        SWAB    R1              ;
1053 004110  005300                        DEC     R0              ; Point at the MSB of the link
1054 004112  010037  177140                MOV     R0,@#RAMADDR    ;
1055 004116  110137  177142                MOVB    R1,@#RAMDATA    ; Write out the MSB
1056 004122  000301                        SWAB    R1              ; Restore R1
1057 004124  005200                        INC     R0              ;
1058                                                               ;
1059 004126  020267  002174                CMP     R2,READFIRST    ; End of linking?
1060 004132  001402                        BEQ     RBL30           ; Yes - then end
1061 004134  010102                        MOV     R1,R2           ;
1062 004136  000740                        BR      RBL25           ; Else loop
1063
1064                               ; Check to see if there are any more frames to be read
1065
1066 004140                         RBL30:
1067 004140  112767  000000  002147        MOVB    #FALSE,NODATA   ; Data was read
1068 004146  016700  002156                MOV     READLAST,R0     ;
1069 004152  062700  000500                ADD     #FRAMELEN,R0    ;
1070 004156  026700  002150                CMP     TOPADDR,R0      ; Top of buffers?
```

```
1072 004162  100002                        BPL     RBL35           ;
1073 004164  012700  001600                MOV     #RXBASE,R0      ;
1074 004170                          RBL35:
1075 004170  005200                        INC     R0              ;
1076 004172  052700  100000                BIS     #100000,R0      ; Read operation to follow
1077 004176  010037  177140                MOV     R0,@#RAMADDR    ; See if the next buffer is full
1078 004202  113701  177142                MOVB    @#RAMDATA,R1    ;
1079 004206  130127  000200                BITB    R1,#B 10000000 ; Is the buffer ready?
1080 004212  001023                        BNE     RBEXIT          ;
1081
1082                                 ; Check the following frame
1083
1084 004214  062700  000500                ADD     #FRAMELEN,R0    ;
1085 004220  042700  100000                BIC     #100000,R0      ;
1086 004224  026700  002102                CMP     TOPADDR,R0      ; Top of buffers?
1087 004230  100002                        BPL     RBL37           ;
1088 004232  012700  001600                MOV     #RXBASE,R0      ;
1089 004236                          RBL37:
1090 004236  005200                        INC     R0              ;
1091 004240  052700  100000                BIS     #100000,R0      ; Read operation to follow
1092 004244  010037  177140                MOV     R0,@#RAMADDR    ; See if the next buffer is full
1093 004250  113701  177142                MOVB    @#RAMDATA,R1    ;
1094 004254  130127  000200                BITB    R1,#B 10000000 ; Is the buffer ready?
1095 004260  001404                        BEQ     RBE01           ;
1096
1097                                 ; Exit point of procedure
1098
1099 004262                          RBEXIT:
1100 004262  112767  000000  002024        MOVB    #FALSE,NOMOREDATA ; Not the last frame
1101 004270  000403                        BR      RBE10           ;
```

```
1103                                    ; Show if there is no data in the next buffer
1104
1105 004272                            RBE01:
1106 004272  112767  000001  002014             MOVB    #TRUE,NOMOREDATA ;
1107 004300                            RBE10:
1108 004300  000207                             RTS     PC               ;
```

```
1110                                    ;=============================================================================
1111                                    ;
1112                                    ;               Interrupt Service Routines
1113                                    ;
1114                                    ;=============================================================================
1115                                    ;
1116                                    ;       1. Line Time Clock Interrupt Vector
1117                                    ;
1118                                    ;-----------------------------------------------------------------------------
1119 004302                            LTCSR:
1120 004302  010146                            MOV     R1,-(SP)                ;
1121
1122 004304  005267  002012                    INC     TIMERCOUNT              ; Increment the general timer
1123 004310  005267  002010                    INC     SCANTCOUNT              ;   and the SCAN time counter
1124 004314  126727  002004  000062            CMPB    SCANTCOUNT,#50.         ; If 1 Sec passed then scan
1125 004322  001016                            BNE     LTCEXIT                 ;
1126
1127                                    ; Set NA to MA+1 to enable new nodes to enter net
1128
1129 004324  010546                            MOV     R5,-(SP)                ;
1130 004326  012705  000001                    MOV     #CR1,R5                 ; If interrupted by WDSR
1131 004332  110537  177144                    MOVB    R5,@#WDREGADDR          ; Point at the CR1
1132 004336  112737  000151  177146            MOVB    #151,@#WDREGDATA        ; Instruct WD to take NAR
1133
1134 004344  012605                            MOV     (SP)+,R5                ;
1135 004346  110537  177144                    MOVB    R5,@#WDREGADDR          ; Point at the old value
1136 004352  112767  000000  001744            MOVB    #0,SCANTCOUNT           ; Zero the counter
1137
1138 004360                            LTCEXIT:
1139 004360  012601                            MOV     (SP)+,R1                ;
1140 004362  000002                            RTI                             ;
```

```
1142                                    ;-------------------------------------------------------------------
1143                                    ;
1144                                    ;       2. Western Digital LAN Controller Interrupt Service Routine
1145                                    ;
1146                                    ;-------------------------------------------------------------------
1147 004364                            WDSR:
1148 004364  112737  000003  177144             MOVB    #IR0,@#WDREGADDR        ; Read the vector
1149 004372  153767  177146  001725             BISB    @#WDREGDATA,IRDATA      ; AND the bits set in the IR0
1150                                                                            ;  register with those in the
1151                                                                            ;  storage space below.
1152 004400  153767  177146  001717             BISB    @#WDREGDATA,IRDATA      ; Must be done twice
1153 004406  110537  177144                     MOVB    R5,@#WDREGADDR          ; Restore pointer to WD reg
1154 004412  000002                             RTI                             ;
```

```
1156                                    ;=================================================================================
1157                                    ;
1158                                    ;                       Messages
1159                                    ;
1160                                    ;---------------------------------------------------------------------------------
1161 004414                            MSG1:
1162 004414    033                              .BYTE   ESC
1163 004415    110                              .ASCII  "H"
1164 004416    033                              .BYTE   ESC
1165 004417    112                              .ASCII  "J"
1166 004420    127     105     114              .ASCII  "WELCOME TO PDPLAN"
1167 004441    000                              .BYTE   0
1168 004442                            MSG10:
1169 004442    015     012     012              .BYTE   CR,LF,LF,LF
1170 004446    124     105     123              .ASCII  "TESTING FOR DUPLICATE ADDRESS"
1171 004503    015     012                      .BYTE   CR,LF
1172 004505    000                              .BYTE   0
1173 004506                            MSG11:
1174 004506    015     012     012              .BYTE   CR,LF,LF,LF
1175 004512    116     105     124              .ASCII  "NETWORK DEAD - BEGINNING INITIALISATION OF NETWORK"
1176 004574    015     012                      .BYTE   CR,LF
1177 004576    000                              .BYTE   0
1178 004577                            MSG2:
1179 004577    116     117     040              .ASCII  "NO ADDRESS DUPLICATION - ENTERING LOOP"
1180 004645    015     012                      .BYTE   CR,LF
1181 004647    000                              .BYTE   0
1182 004650                            MSG3:
1183 004650    101     104     104              .ASCII  "ADDRESS DUPLICATION - CHANGE MY ADDRESS"
1184 004717    015     012                      .BYTE   CR,LF
1185 004721    000                              .BYTE   0
```

```
1187 004722                            MSG4:
1188 004722     015     012                    .BYTE   CR,LF
1189 004724     122     105     101            .ASCII  "READY FOR DATA EXCHANGE"
1190 004753     015     012                    .BYTE   CR,LF
1191 004755     000                            .BYTE   0
1192 004756                            MSG5:
1193 004756     015     012                    .BYTE   CR,LF
1194 004760     124     117     040            .ASCII  "TO WHICH TERMINAL MUST THE MESSAGE BE SENT?"
1195 005033     015     012     012            .BYTE   CR,LF,LF
1196 005036     060     061     110            .ASCII  "01H TO FEH - TO SEND TO A SPECIFIC TERMINAL"
1197 005111     015     012                    .BYTE   CR,LF
1198 005113     106     106     110            .ASCII  "FFH        - TO BROADCAST"
1199 005144     015     012                    .BYTE   CR,LF
1200 005146     040     040     040            .ASCII  "      ENTER THE HEX VALUE > "
1201 005202     000                            .BYTE   0
1202 005203                            MSG6:
1203 005203     015     012     012            .BYTE   CR,LF,LF
1204 005206     116     117     127            .ASCII  "NOW  ENTER THE MESSAGE TO BE SENT (Z TO TERMINATE): "
1205 005273     015     012                    .BYTE   CR,LF
1206 005275     000                            .BYTE   0
1207 005276                            MSG65:
1208 005276     015     012                    .BYTE   CR,LF
1209 005300     101     116     040            .ASCII  "AN ERROR OCCURED IN ENTERING THE ADDRESS, PLEASE RE-ENTER "
1210 005372     050     060     061            .ASCII  "(01H-0FFH) "
1211 005405     000                            .BYTE   0
```

```
1213 005406                            MSG7:
1214 005406     015     012     012            .BYTE   CR,LF,LF
1215 005411     124     131     120            .ASCII  "TYPE: T TO TRANSMIT A FRAME"
1216 005444     015     012                    .BYTE   CR,LF
1217 005446     040     040     040            .ASCII  "      R TO RECEIVE A FRAME "
1218 005501     015     012                    .BYTE   CR,LF
1219 005503     040     040     040            .ASCII  "      > "
1220 005513     000                            .BYTE   0
1221 005514                            MSG8:
1222 005514     015     012                    .BYTE   CR,LF
1223 005516     110     111     124            .ASCII  "HIT ANY KEY TO EXIT"
1224 005541     000                            .BYTE   0
1225 005542                            MSG9:
1226 005542     015     012                    .BYTE   CR,LF
1227 005544     115     105     123            .ASCII  "MESSAGE RECEIVED FROM NODE "
1228 005577     060     060     110    MSG9I:  .ASCII  "00H :"
1229 005604     015     012                    .BYTE   CR,LF
1230 005606     000                            .BYTE   0
1231 005607                            DELMSG:
1232 005607     010     040     010            .BYTE   BS,SPC,BS,NUL
1233                                   .EVEN
```

```
1235                                ;=============================================================
1236                                ;
1237                                ;               Data Area
1238                                ;
1239                                ;=============================================================
1240 005614                        INBUF:          .BLKB   FRAMELEN ; Save space for Message from net
1241         006314'                ENDBUF  =.
1242 006314     000                NOMOREDATA:     .BYTE   FALSE   ; End of Rx buffer chain flag
1243 006315                        NODATA:         .BLKB   1       ; Data read flag
1244 006316     002                TXBUF:          .BYTE   2       ; Transmit buffer counter
1245 006317     000                LONG:           .BYTE   FALSE   ; Long frame flag
1246 006320                        CONTTYPE:       .BLKB   1       ; IEEE Control Byte
1247 006321                        DESTADDR:       .BLKB   1       ; Destination address
1248 006322                        TIMERCOUNT:     .BLKW   1       ; Save space for the timer counter
1249 006324     000                SCANTCOUNT:     .BYTE   0       ; Save space for the scan timer
1250 006325     000                IRDATA:         .BYTE   0       ; Store for Interrupt flags
1251                                                .EVEN           ;
1252 006326                        READFIRST:      .BLKW   1       ; Save space for first buffer pointer
1253 006330                        READLAST:       .BLKW   1       ; Save space for next buffer pointer
1254 006332                        TOPADDR:        .BLKW   1       ; Top of transmit buffers
1255 006334  000377                RBUF:           .WORD   255.    ; MAXNO - the maximum number of characters to
1256                                                                ;         entered
1257 006336                        RBUFD:          .BLKW   1       ; NOENT - the number of characters entered
```

```
1259                            ;=============================================================
1260                            ;
1261                            ;               Stack  Area
1262                            ;
1263                            ;=============================================================
1264                            ;
1265                                    .EVEN
1266 006340                            .BLKW   100.
1267 006650                    STACK:
1268                            ;=============================================================
1269                            ;
1270                            ;               Input buffer area
1271                            ;
1272                            ;=============================================================
1273                            ;
1274 006650                    RBUFI:  .BLKW   1       ; The rest of the memory is for inputs
1275                            ;
```

SYMBOL TABLE

```
ACKRES= 000020      ESC   = 000033      LF    = 000012      NODATA  006315R     RML10   000726R
ADL10   002416R     ETX   = 000032      LONG    006317R     NOMORE  006314R     RML25   000736R
ADL15   002460R     FALSE = 000000      LTCEXI  004360R     NUL   = 000000      RML30   000752R
ADL20   002436R     FILTXB  002644R     LTCSR   004302R     RAMADD= 177140      RXBASE= 001600
AHOLT = 000015      FRAMEL= 000500      MA    = 000017      RAMDAT= 177142      SA    = 000001
BASE  = 177140      FTXEX   003304R     MAIN    000500R     RBEXIT  004262R     SCANFC= 000040
BS    = 000010      FTXL0   002650R     MAXFRA= 000005      RBE01   004272R     SCANTC  006324R
BSIZE = 000004      FTXL01  002674R     ML10    000510R     RBE10   004300R     SNDMSG  001024R
CBLOCK= 000000      FTXL05  002706R     ML12    000520R     RBL01   003346R     SPC   = 000040
CBPH  = 000012      FTXL10  002762R     ML121   000572R     RBL03   003420R     SR0   = 000002
CBPL  = 000013      FTXL20  002770R     ML13    000616R     RBL05   003460R     SR1   = 000004
CIDEL   001222R     FTXL25  003032R     ML14    000624R     RBL07   003566R     SR2   = 000005
CIEXIT  001242R     FTXL30  003046R     ML15    000666R     RBL10   003572R     STACK   006650R
CIL05   001134R     FTXL40  003120R     ML20    000674R     RBL11   003606R     TA    = 000010
CIL10   001134R     FTXL50  003152R     ML30    000710R     RBL15   003654R     TD    = 000011
CIL20   001172R     FTXL55  003162R     MOEXIT  001600R     RBL17   003660R     TIMERC  006322R
CIN     001120R     FTXL60  003240R     MSGOUT  001554R     RBL20   003672R     TIMERN= 000036
CNTRLZ= 000032      FTXL65  003262R     MSG1    004414R     RBL201  003716R     TOPADD  006332R
CONTTY  006320R     IICSA   001456R     MSG10   004442R     RBL21   003754R     TRUE  = 000001
CR    = 000015      IIL10   001512R     MSG11   004506R     RBL22   003776R     TXBUF   006316R
CR0   = 000000      IIL20   001542R     MSG2    004577R     RBL25   004040R     TXFCB = 000000
CR1   = 000001      IL01    001672R     MSG3    004650R     RBL27   004060R     TXLT  = 000016
CTR0  = 000006      IL10    001714R     MSG4    004722R     RBL30   004140R     UAERRO  001406R
CYSKIP= 000000      IL20    001766R     MSG5    004756R     RBL35   004170R     UAEXIT  001400R
DACEXI  001450R     INBUF   005614R     MSG6    005203R     RBL37   004236R     UA10    001302R
DEL   = 000177      INITNE  001602R     MSG65   005276R     RBUF    006334R     UA20    001324R
DELAY   001262R     INS   = 000040      MSG7    005406R     RBUFD   006336R     UA30    001344R
DELMSG  005607R     IRDATA  006325R     MSG8    005514R     RBUFI   006650R     UA40    001354R
DESTAD  006321R     IREC  = 000010      MSG9    005542R     RBUF1 = 177562      UI    = 000003
DL10    001270R     IROR  = 000100      MSG9I   005577R     RCSR1 = 177560      UNASCI  001276R
DML10   002312R     IR0   = 000003      MYID  = 000001      READFI  006326R     WDREGA= 177144
DOASCI  001436R     ITA   = 000002      NA    = 000007      READLA  006330R     WDREGD= 177146
DUMMYL  002262R     ITD   = 000001      NAR   = 000014      READWD  003324R     WDSR    004364R
```

```
ENDBUF= 006314R         ITERR = 000200         NDL05   002502R         RECCHA= 000001         WTXFCB= 000200
ENQ   = 000005          ITOK  = 000004         NDL10   002546R         RECMSG  000716R        XBUF1 = 177566
EOT   = 000032          ITRAN = 000020         NETDEA  002472R         RMEXIT  001022R        XCSR1 = 177564


. ABS.  000204     000
        006652     001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  8247 WORDS  ( 33 PAGES)
DYNAMIC MEMORY:  9820 WORDS  ( 37 PAGES)
ELAPSED TIME:  00:00:48
```

APPENDIX I

Program Users' Guides

I.1 Using EMS

To use this program proceed as follows. Make a copy of the EPROMs onto the EMS disk. (The EMS disk presently contains a copy of the EPROMs used in the micro-computers.) The files must be named EPDIR.COM for the first one and then EP1.COM, EP2.COM and so on. To run the program type:

EMS<CR>

This will give the main menu. If M is chosen, a display of the program map is sent to the console. If D is selected then the user is prompted for the name of the file to delete. The EMS program will either delete the program specified or, if there is no such program in the directory, it will inform the user. If the program is deleted, the user will be instructed to re-program the EPDIR EPROM with the contents of EPDIR.COM which is now the updated version of the directory. If A is chosen, then the user will again be prompted for the name of the file. Should the specified program already be present on the pseudo-disk, then the user is given the choice of replacing the old program or aborting the addition utility. The EMS program will then load the program from the specified file to the EPn.COM file, giving the user a list of the EPROMs that need reprogramming. Typing a R will return the user to CP/M.

I.2 Using the TAC User Routines

The user routines may be used by any 8085 program run on the SABus computer. The program must first be loaded from the EPROM card by typing:

*LAN<CR>

to which there are two possible responses. The first is the duplicated address message:

Duplicate TAC addresses

In this case the EPROM's have been programmed with the incorrect TAC address value. This address is found at address 0025H and should be different for each TAC. The second possible response is:

TAC initiated

This may take a while to appear. If no other TAC is on the network, it will wait until another one is initialised. This message means, therefore, that the TAC has been initialised and that it is currently taking part in a logical ring.

All the entry points are entered by jumping to the appropriate offsets with the registers set to the required values. To enable direct control of the WD2840, the registers may be accessed directly by the 8085 IN and OUT commands. The TAC is situated at address 70H in the I/O page of the 8085 but is mini-jump selectable to any address on a 16 byte boundary.

I.3 Using the CP/NET Programs

The TAC User routines must first be loaded as explained above and then the CP/NET programs may be loaded from the EPROM card when CP/NET is run from the SABus computers by typing:

*CPNET<CR>

The rest of CP/NET is as specified in the CP/NET Users guide.

I.4 Using the PDPLAN Program

To use this program it is necessary to load the program down from the PDP to the FALCON computer. This is done in two stages. The first section is to load down, under ODT control, a loader program which will enable the fast transmission of the user program. This loader program will read the start address of the block and the length of the block from the first two words received and then write the subsequent data to the correct place in memory.

There are two programs which run on the PDP 11/23 and which are used to send the programs. The first, PDPLD, will send the data in a way that ODT would expect the user to type it in. The second will start the loader program prevoiusly sent and then write the user program into memory.

To run these programs, set the line connected to the FALCON to NOECHO and SLAVE mode, and then type:

RUN PDPLD<CR>

on another terminal logged on under 100,7. This program will prompt the user for the file name and the terminal of the FALCON. The program name should be LD.OBJ. Once this is complete, the user should then type:

RUN DNL<CR>

to run the download program. This too will prompt for a file name and the destination terminal. The file would be PDPLAN.OBJ. Once this is complete, the user may run the PDPLAN program by typing

500G

on FALCON. The response will be one of two messages. Firstly, if the network is active then:

WELCOME TO PDPLAN

TESTING FOR DUPLICATE ADDRESS

will be displayed. If there is a duplicate address then the following message will appear:

ADDRESS DUPLICATION - CHANGE MY ADDRESS

and the program will stop. If no address duplication occurs then the following messages appear:

```
NO ADDRESS DUPLICATION - ENTERING LOOP

READY FOR DATA EXCHANGE


TYPE: T TO TRANSMIT A FRAME
      R TO RECEIVE A FRAME
      >
```

The TAC is now in the logical loop and ready for data exchange. The user may now transmit a frame by choosing T, in which case the user will be prompted for a destination address by the following message:

```
TO WHICH TERMINAL MUST THE MESSAGE BE SENT?

Ø1H TO FEH - TO SEND TO A SPECIFIC TERMINAL
FFH        - TO BROADCAST
      ENTER THE HEX VALUE >
```

Once the hex value has been entered, which may be corrected by the <DEL> key, the user will be informed that it was incorrect if that is the case, and will be prompted for a new value or, if the address was valid, then the user is prompted for the message with the following message:

```
NOW ENTER THE MESSAGE TO BE SENT (Z TO TERMINATE):
```

The Falcon will then accept up to 65Ø characters which will be sent. Once the console displays the transmit or receive message again it means that the frame has been sent.

To receive a message, the user may type R to the transmit or receive message, but each time the TAC receives a message it will automatically be displayed.

Appendix J

A detailed discussion on the choice of the U.C.T. LAN architecture.

(This is a copy of Chapter 2 of a previous draft.)

CHAPTER 2

The U.C.T. LAN Architecture

No one LAN architecture is the best for all applications. It is thus necessary to carefully examine each network's requirements and design a LAN specifically for that application.

This chapter attempts to show that the Token-passing bus LAN was the best architecture for the U.C.T. LAN.

2.1 The U.C.T. LAN Requirements

The requirements for this LAN are as follows:

1. Low cost.
2. Minimal cabling.
3. A maximum of 30 stations.
4. 1 Mbps transmission rate.
5. An overall physical path of not more than 800 metres.

Any one of a large number of architectures would fulfil these requirements, e.g. slotted-ring, CSMA/CD bus, token-passing rings or busses and a variety of star networks. In order to provide the most cost-effective and long-term design, Wittlin and Ratner [27] state that one should choose an architecture that adheres to some standard. This ensures that for future additions and modifications there are a number of possible equipment suppliers, thereby guaranteeing the availability of new equipment at (hopefully) reasonable prices.

The choice of the architecture was thus limited to one of those specified by the IEEE 802 standards which are generally considered to be the emerging LAN standards.

2.2 Overview of the IEEE 802 Architectures

Up to the beginning of this decade computer networks were most often designed around the International Standards Organisation's Open Systems Interconnection Reference Model (ISO/DIS 7498), known as the ISO RM, which defines computer intercommunication. With the increased demand for computer communication caused by the development of LSI technology and the subsequent creation of the micro-processor, manufacturers began designing local computer communications networks based on their interpretation of the ISO RM. This

led to a proliferation of LAN architectures. The IEEE appointed the 802 committee with the task of attempting to standardize on certain LAN architectures. The results of their work is a suite of standards collectively called the IEEE 802 standards.

The general aim of the 802 standards was to transform the ISO RM into a model that applies directly to local area networks. This was achieved by re-defining the lowest two ISO RM layers, the Physical and Data-Link layers as the Physical (PHY), Media Access Control (MAC) and Logical Link Control (LLC) layers. The PHY layer controls the physical signalling over the network, the MAC layer controls the access to the shared medium, and the LLC layer is in charge of the logical transmission of data over the link.

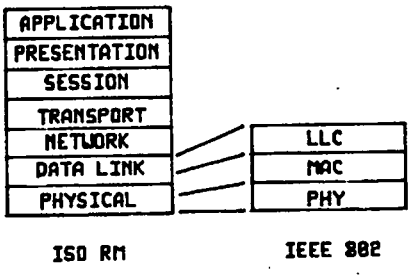


Figure 2.1 - The ISO Model and the IEEE 802 Model

Very early in its existence the IEEE 802 committee realised that it was unable to standardize on a single LAN architecture. This led to the formation of a number of sub-committees to examine the different networks. By the

beginning of 1985 standards defining four architectures had been drafted, namely:

1. ANSI/IEEE 802.3 - 1985 defining a baseband CSMA/CD bus network
2. ANSI/IEEE 802.4 - 1985 defining a broadband and
3. baseband Token-passing bus network
4. ANSI/IEEE 802.5 - 1985 defining a baseband Token-passing ring network

2.2.1 IEEE 802.3 - CSMA/CD Bus Network

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a technique which enables a number of nodes to transmit information (at random intervals) over a common medium. Random access is achieved, firstly, by sensing the network for activity (carrier sensing) before transmitting and, secondly, once the network is found to be unused and the station has begun its transmission, by providing a "listen-while-talk" facility (collision detection) which compares the data being sent to that on the network. Inequality between these signals indicates a frame collision, caused by more than one station transmitting simultaneously.

To allow time for the receiver to process the frame and

to wait for the dissipation of any transients on the network, the IEEE 802.3 standard defines a period of quiet between frame transmissions called the "inter-frame" delay. To enforce this period, the station must continuously sense the network and begin a timer on detection of no transmission, i.e. the end of a frame. This prevents the transmitter from attempting to access to the network before the end of the inter-frame period.

When data is presented to the MAC layer for transmission, it constructs a frame by adding the header fields, checksum and flags, and places the frame into a transmitter queue. Transmission of these frames is postponed if another station is using the network or if the inter-frame period has not elapsed since the previous transmission. Once this period has expired, transmission is initiated. Because no station will begin its transmission whilst the network is in use, a collision must occur within the end-to-end propagation time for the network and thus be detected at the transmitting stations before the end of two such propagation times. Collision detection is also the reason for specifying a minimum frame length to be longer than twice the end-to-end propagation time for the network.

When a collision does occur, the transmitting station "jams" the network by issuing a number of jamming bits (48

in the IEEE 802.3 case) thus ensuring that all participating stations are aware of the collision. Jamming is required because the analogue collision detection circuitry may not detect a very short collision, e.g. if a second station began its transmission momentarily before receiving the first station's frame and instantly stopped on detection of the collision.

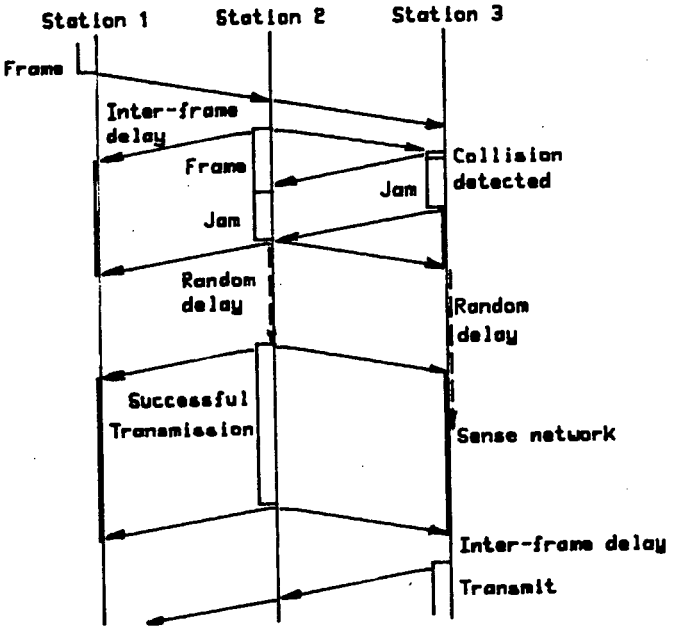


Figure 2.2 - CSMA/CD Bus Operation

After the detection of a collision, contention between the responsible stations is resolved by delaying their re-transmission attempts by a random number (r) of "slot times". This slot time is the maximum time required for all the stations on the network to detect a collision and must be greater than twice the network's end-to-end propagation time plus the network jamming time. It is defined as 512

bit times by the IEEE.

The random period (**r**) is calculated by a controlled randomisation process called "truncated binary exponential backoff", and is an integer which lies between 0 and 2^{k}, where **k** is the smaller of the retransmission count and 10. A station will attempt to re-transmit the frame a maximum of 16 times after which it will inform the station management and terminate. Only once a frame has been successfully transmitted, will a new frame be sent.

The IEEE 802.3 standard defines the following parameter values:

1. Slot time = 512 bit times
2. Inter-frame gap = 9.6 usec
3. Jamming bits = 48
4. Minimum frame size = 64 octets (512 bits)
5. Maximum frame size = 1 518 octets
6. Maximum end-to-end distance = 2.5 km
7. Data Rate = 10 Mbps.

2.2.2 IEEE 802.4 - Token-bus Network

Token-passing is a distributed polling access method used to provide controlled sharing of a common medium by a

number of terminal devices.

Essentially the Token-passing operation is this: the right to the network is passed from station to station around a logical ring on the bus network in the form of a unique control frame called the "token". The station that is addressed by the token may transmit up to a pre-defined maximum number of frames or simply pass the token to its successor. Token rotation is in descending order of station addresses, except that the lowest station passes the token to the station with the highest active address in order to complete the logical ring. Each station keeps a record of its own address, its successor's address and its predecessor's address. By fixing the rotation of the token (to either descending order, as in the IEEE case, or ascending order which the WD2840 uses), the quick inclusion of new stations and the re-establishment of the logical ring after failure, is facilitated.

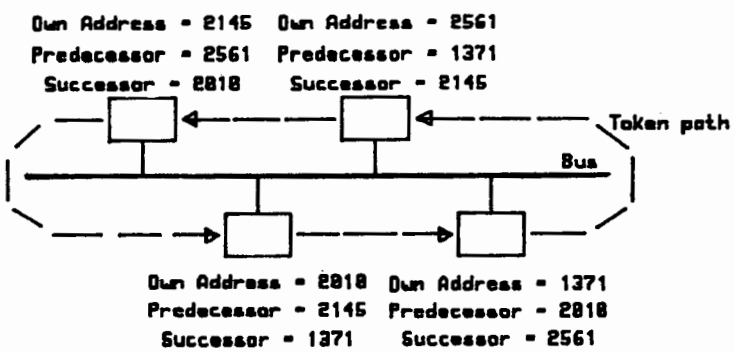


Figure 2.3 - The Logical Ring on a Token-bus Network

To prevent a single station transmitting for a long period of time ("network hogging"), each station keeps a record of the time that it has held the token. By comparing this value to a fixed maximum token hold time, a station may determine if there is enough time to transmit a queued frame or if it must pass the token.

The IEEE 802.4 standard also defines an optional priority structure which consists of 8 priority levels or service classes. The top four, service classes 4 to 7, are for "synchronous data" which require a guaranteed transmission time, while frames of the lower 4 service classes, 0 to 3, use the residual bandwidth. The service classes are paired to form 4 access classes, where service classes 0 and 1 form access class 0, 2 and 3 form access class 2, 4 and 5 form access class 4 and service classes 6 and 7 form access class 6. Each access class is given a "target" token rotation time; if the token returns after passing around the logical ring and the target time has not been exceeded, a frame from the particular service class's queue may be sent. Thus packets may be sent at the same access class, i.e. have the same target rotation time, but the LLC layer may still define which packets must be sent first, by placing them in the higher service class's queue.

The highest access class, class 6, uses the maximum

token hold timer to determine if it is permitted to send a frame. Should all the frames from service class 7 and 6 be sent and should time still be available on the maximum token hold timer, the token will be passed within the station, to the next service class. The new service class will determine if the token rotation time is less than its target value and, if so, transmit one or more queued frames, but after each frame ensure that there is time available on the target rotation timer. The token will pass to the next service class only once transmission of all queued frames is complete or if the target token rotation time has been exceeded. This continues down to the lowest service class after which the token will be passed to the next station. (A station not using the priority structure, will transmit frames with an access class of 6, and so use only the maximum hold timer.)

Figure 2.4 below illustrates this structure by way of an example. The station under consideration has three service classes, 7, 3 and 1, which fall into access classes 6, 2 and 0 respectively. The maximum hold time is set to, say, 1000 octet times. The target rotation time (TRT) for access class 2 is 2000 and access class 0 is 1500. The station has two frames of 200 octets queued for service class 7, one frame of 500 octets for service class 3 and one frame of 128 octets for service class 1. When the token is

received after passing around the logical ring, it has taken 1000 octet times. The two highest priority frames are sent, leaving 600 octet times on the maximum token hold timer for the lower levels. The token is thus passed to the service class 3 sub-station, which calculates that there are 600 octet times available for transmission based on its TRT (i.e. 2000 - (1000 + 400)), and so it transmits its frame. Only 900 (i.e. 400 + 500) octet times have been used since the token was recieved, which is within the maximum token hold time, thus the token is passed to level 1. This level discovers that its target rotation time has been exceeded (i.e. 1500 < (1000 + 400 + 500)). The token is therefore passed to the next station without the transmission of any level 1 frames.

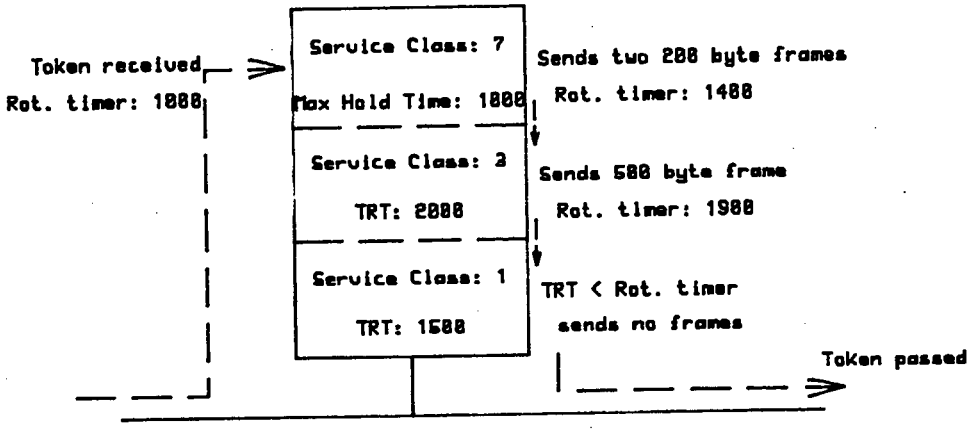


Figure 2.4 - Priority Transmission on the Token-bus

This priority frame handling structure makes the IEEE Token-bus popular for voice and control applications where

large delays cannot be tolerated. Such frames can make use of the "synchronous" data transmission.

In a real network there are a number of conditions which require special procedures for their handling. These conditions are: new station inclusion, initialization of the logical ring, action on discovery of a lost token and station removal.

(a) New Station Insertion:

A new station may only be included in the logical ring when invited to do so by one of the already active stations. This is achieved by having the active stations issue "Solicit Successor" frames at regular intervals. A "Solicit Successor" frame specifies a range of station addresses that may respond (with a "Set Successor" frame) in order to be included in the logical ring. During the successor soliciting phase, the token-holding station attempts to find a new station whose address falls between its own and its successor's addresses. It thus looks for a station with the highest address in the specified range, or, if the soliciting station is the lowest active station, it looks for a new station with lower address or one that has a higher address than the highest active station.

It is possible that more than one station in the address range, specified by the solicit successor frame, will require inclusion in the logical ring. To ensure that only the station with the highest address in the range is included, a contention resolution process, which makes use of "response windows", is defined. A response window is a fixed period, calculated from the completion of a frame to the start of the responding station's frame. Its purpose is to define the starting time of a response frame and thereby enable the sort process defined below.

If the token-holding station receives a garbled message, it will assume that more than one station has responded with a "Set Successor" frame and will thus issue a "Resolve Contention" frame. Those stations which responded to the "Solicit Successor" frame will transmit a second "Set Successor" frame, but this time delayed by 0, 1, 2 or 3 response window periods after the end of the token holder's "Resolve Contention" frame. This delay is computed by complementing the two most significant bits in the station's address, i.e. if the two bits are 00, 01, 10 or 11 the transmission is delayed by 3, 2, 1 or no response windows respectively. A station will only respond if no other station had already begun its transmission.

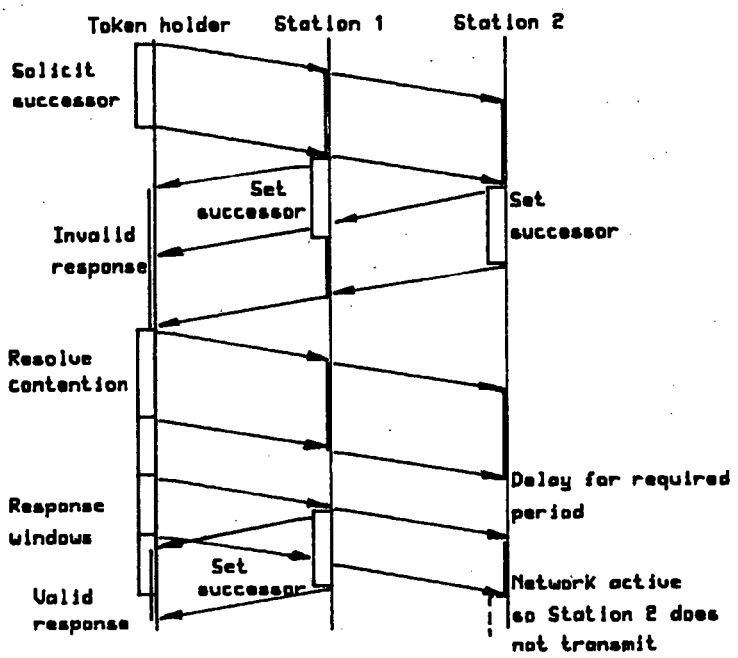


Figure 2.5 - Response Windows

If the token-holding station is still unable to interpret the frame, it will issue a second "Resolve Contention" frame. Only those stations that sent frames in response to the previous "Resolve Contention" frame will respond again, but this time their transmission will be delayed by a value calculated, in the same way, from the second most significant pair of bits in the address. This procedure will continue until a valid "Set Successor" frame (which must be issued by the station with the highest address in the specified range) is received by the token holding station.

The stations excluded by the sort process will be

included in the logical ring by subsequent "Solicit Successor" frames.

(b) Initialization of the Logical Ring:

The initialization process consists of two phases, the first is the selection of a single initializer (as more than one station could attempt to assume this role). The second phase establishes the logical ring on the network using a sort-by-address process described in (a) above.

On network start-up, or after the network has been inactive for 6 or more "slot times", an attempt will be made to (re-)establish the logical loop. The maximum time required for a MAC layer to receive an immediate MAC layer response, i.e. one generated by the remote MAC layer itself and not by its higher levels, is called the "Slot time" (T_s), which is equal to one "Response Window", and is defined by IEEE as follows:

$$T_s = INT\left[\frac{\{2(T_p+T_{sta})+T_{sm}\}/T_{sym}+7}{8}\right] \text{ [Octet times]}$$

Where T_p is the end-to-end propagation time for the network.

T_{sta} is the time between the data being ready and the start of transmission.

T_{sm} is the safety margin (at least 1 symbol time long).

T_{sym} is the symbol time. (1 μs for a 1 MHz network)

On start-up each station has its "Inactivity Timer" set to 7 slot times. Once the logical ring has been established, the node with the lowest address resets its value to 6 slot times.

The expiry of the "Inactivity Timer" prompts the station to claim the token by transmitting a "Claim Token" frame. Assuming that more than one station transmitted such a frame, which could occur at network start-up or if the lowest station failed, the IEEE 802.4 standard provides a means of isolating a single station, namely the highest station that sent a "Claim Token" frame.

This contention resolution process, which is a sort based on the length of the "Claim Token" frame, is achieved by each potential initializer setting the length of its "Claim Token" frame's data field to reflect the value of a pair of bits in the address. The first "Claim Token" frame's data field will be 0, 2, 4 or 6 slot times long

depending on the value of most significant pair of bits in the address, i.e. if the values of the top two bits were 00, 01, 10 and 11, the data field would be 0, 2, 4 and 6 slot times long respectively. Once the frame has been sent, the station will wait one slot time (for the propagation of its own frame and other station's frames, sent at the same time) and then sense the network for activity. Only those initializers that sense no activity are allowed to continue vying for initializer status; the other stations must remain silent.

The contention resolution process continues by sending a second "Claim Token" frame. This time the length is related, in a similar way, to the second most significant pair of bits in the address. This procedure continues until all the bits in the address have been used, as well as two random bits. (The random bits allow for the error condition where two stations have the same address.) The station which sends the last "Claim Token" frame and hears nothing has the token and assumes the role of initializer.

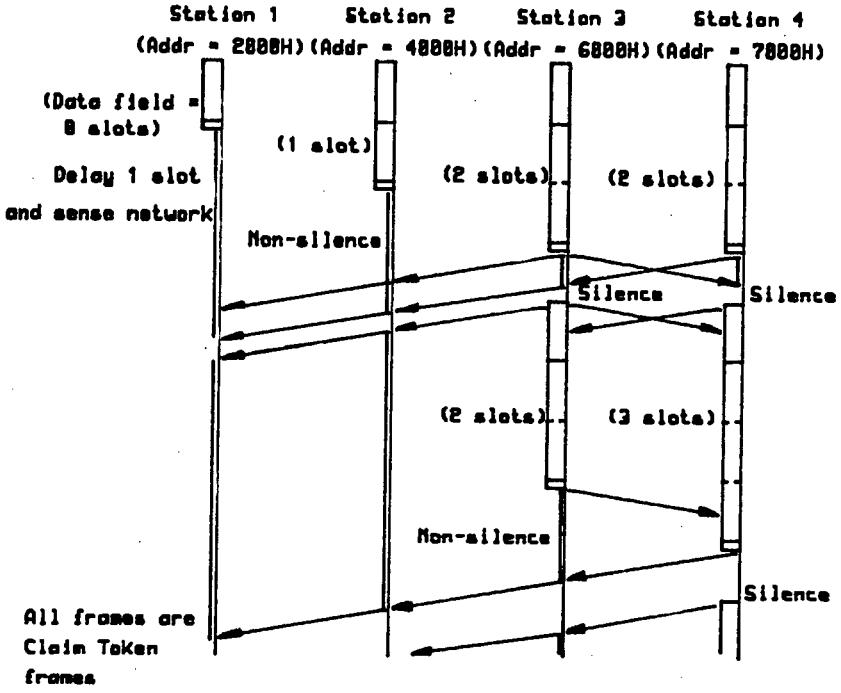


Figure 2.6 - The Contention Resolution Process

The second phase, that of establishing the logical ring, is achieved by the sort-by-address process described in (a) above.

(c) Station Removal or Failure:

A station passing the token is responsible for ensuring that network activity follows the transmission of the token. No activity prompts the re-transmission of the token. If there is still no response after the second token transmission the token-holding station must attempt to find the next lower station by issuing a "Who Follows" frame

which is an attempt to make contact with the failed station's successor. This frame requests the station whose predecessor address is equal to the value in the data field to respond with a "Set Successor" frame. If such a station is active on the network, which will be the case unless both of the two subsequent stations have failed, it will respond with a "Set Successor" frame and the ring will be re-established.

No response to a second "Who Follows" frame will lead to the issuing of a "Solicit Successor" frame in order to find any other active stations. If a response is received from one or more stations, the logical ring is formed as described in (a) above. A silent network after two attempts means that either the station has become deaf or the network is unusable, and so the token holding station will enter the listen-only state.

If a station fails while holding the token, the lost token recovery procedure (described in (d) below) will be invoked.

To permit orderly exclusion of a station from the logical ring, a station may send a "Set Successor" frame, containing its successor's address, to its predecessor. On the next token rotation the station will be excluded.

(d) Token Loss and Duplicate Token:

If the Token is lost through a station failure, recovery will be initiated by the expiry of an inactivity timer. The recovery procedure follows the strategy described in (b) above.

If a station which has the token detects a transmission that is not a response frame, it will assume that there is a duplicate token on the network. Under this condition it will not pass the token and will wait for a new one before transmitting any further frames.

2.2.3 IEEE 802.5 - Token Access Ring

In the Token-ring network, each station is connected to two others by means of a point-to-point link to form a closed ring. This connection is made by means of a Network Interface Unit (NIU) which ensures that the network is not halted should a station become inactive.

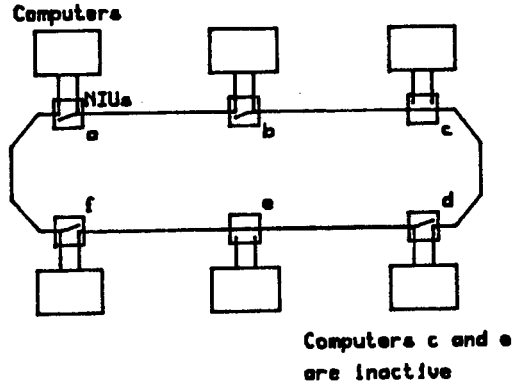


Figure 2.7 - The Token-ring Physical Structure

Frames are transmitted around the ring by being relayed from one node to the next. To ensure ordered access to the common medium, a Token frame is also circulated. A station, which has data to transmit, gains access to the network by setting the busy/free bit in a passing token, thereby converting the token into a start-of-frame marker to which the frame is appended. Because the IEEE 802.5 Token-ring is a single token protocol, i.e. only one token may be present on the network at any time, it is the responsibility of the token holding station to issue a new token only once the frame header has circumnavigated the network, providing that the station has no more queued data or it has sent the maximum number of frames per token reception.

A priority facility, incorporating 7 priority levels (1 - 7), is provided on the Token Ring. When a station receives a high-priority frame from its LLC layer it will

transmit the frame on receipt of the next free token of lower or equal priority. If the network is busy, the station will set the appropriate "reservation" bits in the header of a passing frame, indicating the level of priority required. On receipt of this header, the token holding station will issue a high-priority token, allowing access to frames of that or a higher priority only. In order to prevent a high-priority token from circulating continuously, the station that first issued the high-priority token is responsible for returning it to the priority it was received at, after a single rotation.

Frame reception is achieved by the NIU copying all the passing frames into an internal buffer. The destination address is then compared with its own address and if it corresponds the frame is passed to the computer via the LLC layer.

Ring maintenance is achieved by assigning to one of the active stations the task of "Monitor". On start-up each station enters stand-by monitor mode. After a period of network inactivity, defined by the "Inactivity Timer", any station may issue a "Claim Token" frame in an attempt to become the active monitor. To ensure that only one monitor exists, a station vying for monitor status defers to a station with a higher address. The receipt of a "Claim

Token" frame with its own address as the source address means that a station has been successful in its attempt to become the Monitor. (A "Claim Token" frame with a lower source address than the stations own address is ignored.) If an active monitor receives a "Claim Token" frame, an error has occured. To facilitate quick recovery, it will cease to be the monitor and return to stand-by status.

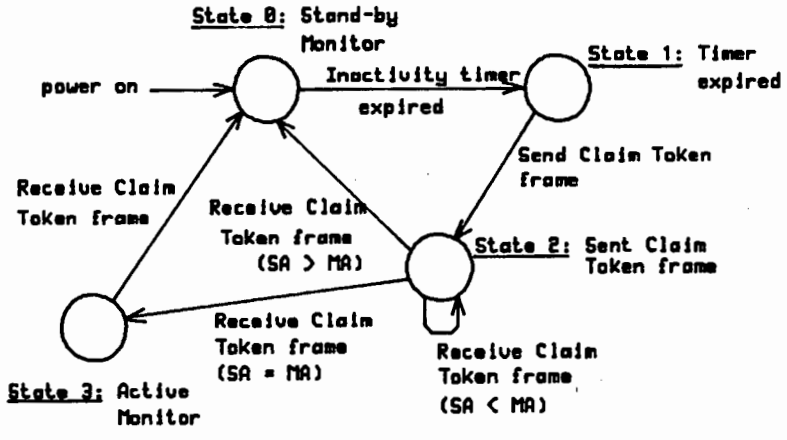


Figure 2.8 - Simplified Token-ring Monitor State Diagram

The Monitor station has the following tasks:

1. It must issue a new token when the network has been inactive for a specified period.
2. It must check for and recover from continuously rotating high-priority tokens
3. It must prevent the continuous rotation of a frame.

2.2.4 The WD2840 Token-bus

The Token-bus has often not been considered as a viable network architecture because of the complexity of its MAC layer. Recently, though, Western Digital Incorporated introduced the WD2840 Token-access controller which is a single chip that implements the Token-bus MAC layer. The WD2840 was used in this project, but unfortunately its implementation of the MAC layer differs from the IEEE 802.4 specification in several respects, namely:

1. Direction of token rotation.
2. Frame format.
3. Dynamic operation.

1. Direction of Token Rotation:

The direction of the token rotation is opposite to that of the IEEE 802.4 standard, in that the token is passed from a station with a lower address to one of with a higher address. The last station, with the highest address, will then pass the token back to the first station, i.e. the one with the lowest address, and so complete the logical ring.

2. Frame Format:

Firstly, the fields for both the destination and source addresses are only 1 octet long, and not 2 or 6 as defined by IEEE.

Secondly, the frame check sequence is a 16-bit field rather than a 32-bit field.

Thirdly, all frames contain an access control field. This field performs two functions: either it requests an immediate acknowledgement to the frame being sent (when it has a value of 255) or it "piggy-backs" the token by containing a valid address, i.e. between 1 and 254. (A value of 0 has no meaning or effect.) A data frame can therefore simultaneously transfer data and pass the token to the same or different stations.

The shorter address field and FCS reduce frame overhead, thereby improving the potential throughput. These fields limit the WD2840 to non-IEEE 802.4 networks and, because of the shorter FCS, its reliability is lower.

3. The Dynamic Operation:

(a) Initialisation of the Logical Ring:

When the inactivity timer (or the Network Dead timer) expires and the station has been assigned the task of initializer, the station will assume that the token has been lost or that the network has just been powered up and will attempt to (re-)establish the logical ring.

On network start-up, the first station that is powered up will attempt to establish the logical ring. This is achieved by a "scan operation" where each station sets its successor's address value to its own address plus one. An attempt is then made to pass the token. If no response is forthcoming, the station will increase the value and attempt transmission again. Once a response is evoked, the station will save the address as that of its new successor. As each station receives the token it will scan for its successor until the initialising station again receives the token.

(b) Token Loss:

Recovery from a lost token is achieved as follows: When the token holding station passes the token it listens for a response. If nothing is received in the required response period, it re-transmits the token. If there is still no response it will scan for a new station, using the procedure in (a) above.

If a station fails while holding the token, after a period of time defined by the shortest Network Dead timer, a station will set its successor address to its own address plus one and begin the scanning procedure as described in (a) above.

(c) Station Insertion:

The WD2840 controller has no means of calculating when to solicit for a new successor. It is thus the responsibility of the host computer or the interface hardware to initiate the scan cycle. This is achieved by setting the station's successor address to its own address plus one. On the next token pass it will search for a

successor. A new station, having an address which falls between the station's own address and its successor's address, will thus be included in the logical ring.

(d) Station Removal or Failure:

The WD2840 does not define a "Set Successor" frame as it keeps no record of its predecessor's address. A station will thus be removed after ignoring the next token, resulting in a scan for the new successor by its predecessor.

If the station fails while holding the token, the expiry of a Network Dead timer will invoke the logical ring re-establishment procedure described in (a) above.

2.3 A Comparison of the Architectures

In this section the IEEE architectures and a Token-bus network using the WD2840 TAC are discussed and compared in terms of performance, reliability and physical considerations.

2.3.1 Performance

The performance of a LAN is measured in terms of throughput and queueing delay. The throughput is the rate at which user data is transported across the LAN, while the queueing delay is the time between the data being presented to the first station's NIU and its presentation to the destination node.

(a) Throughput:

The IEEE 802.3 CSMA/CD Bus Throughput:

The distributed and random nature of CSMA/CD means that there is a finite probability that more than one station will attempt to access the common medium at any particular time. The maximum throughput of this network is calculated assuming that all the active stations will always have data to send, i.e. a new frame is immediately queued once a frame has been sent. After the successful transmission of a frame, all the active stations will wait the required inter-frame period and then begin transmitting. Should more than one station be active, a collision will occur. Frame re-transmissions are then controlled by the truncated binary exponential backoff method described in the previous section.

To formulate the throughput efficiency, it is assumed that the probability of any one station transmitting immediately after a collision or successful transmission is P and the probability of that station not transmitting in the first slot is $\{1-P\}$. The probability of all Q active stations transmitting immediately after the collision is therefore P^Q, and the probability of all stations deferring transmission is $\{1-P\}^Q$. The probability of all but one station deferring transmission is:

$$x = Q\,P\{1-P\}^{(Q-1)} \qquad \ldots (1)$$

The truncated binary exponential backoff random delay generator attempts to give P a value of Q^{-1}, which in turn maximizes x. The probability of only one out of Q stations transmitting in one slot time is A, and may be defined as:

$$A = x_{max}$$

$$= \left\{1 - \frac{1}{Q}\right\}^{(Q-1)} \qquad \ldots (2)$$

The average number of slots wasted before a successful transmission occurs may now be calculated. The probability of any of the Q active stations transmitting immediately after the collision is simply A. The probability of a successful transmission after a single slot is $A\{1-A\}$. The probability of a successful transmission after i slots is

thus: $A\{\{1-A\}^i\}$. The mean of the sum of all the probabilities, i.e. the sum $A\{\{1-A\}^i\}$ for values of i from 1 to infinity, will be the mean delay between successful transmissions.

This mean, W, is thus the average waiting time (in slot times) and is:

$$W = \sum_{i=1}^{\infty} A\{1-A\}^i$$

$$= \frac{1-A}{A} \qquad \ldots (3)$$

To ensure that the data presented to the MAC layer arrives at its destination correctly, it is placed into a frame. The creation of this frame requires the use of additional bits which in turn reduce the effective throughput.

Two other factors combine to further reduce the efficiency of the network, these being the inter-frame delay and the minimum frame length required by CSMA/CD for collision detection. The inter-frame delay reduces the efficiency because no data transfer can take place during its duration. The minimum frame length does so because, if less data is supplied than is required to fill the minimum

frame, the difference is made up of padding characters.

The throughput efficiency of the CSMA/CD network is thus:

$$S = \frac{T_d}{\{T_d + \{W \bullet T_s + T_p + T_f + T_i\}\}} \quad \ldots (4)$$

Where T_d is the average time taken to send the LLC data.

T_s is the slot time.

T_p is the time spent sending padding characters.

T_f is the time spent sending framing characters.

T_i is the inter-frame delay.

W is the waiting time in slot times defined by (3).

The inter-frame gap and the framing bits are defined for the IEEE 802.3 network. The variables are the packet size and network propagation time. Using equation (4) and the following parameter values, the maximum throughput for the IEEE 802.3 network may be calculated:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.

3. Transmission path = 800 m.
4. Frame length = 256 octets.
5. Frame overhead = 138 bits.
6. Inter-frame delay = 9.6 usec.

The above values allow for a transmission rate of 452.37 frames per second which corresponds to a maximum throughput efficiency of 92.65%. All the other parameter values, with the exception of the frame length, are fixed for the U.C.T. network. Figure 2.9 below illustrates the effect of the value of this parameter on the performance of the CSMA/CD network.

The IEEE 802.4 Token-bus Throughput:

The predominant cause of inefficiency in the Token-bus architecture is due to the rotation of the token around the logical ring. This is clear when one considers that a new token has to be generated and transmitted across the network for each token pass. If the token is **B** bits long, it will take **B/C** seconds to be transmitted from a station onto a network with a capacity **C** bps, and take **B/C** plus the propagation time (T_p) to reach the most distant station. The Token rotation time (T_r), when only the token is circulating and no data is being transmitted, is thus the token transmission time per station multiplied by the number

of active stations on the logical ring (**Q**).

$$T_r = Q\{B/C + T_p\} \quad \ldots (5)$$

If only one station is transmitting while the rest are simply passing the token, there will be a lower throughput because of the token regeneration at each station. It is clear that if more stations are transmitting information, or if more data is allowed per token hold, that the throughput will be proportionally higher.

As in the CSMA/CD case, the frame overhead will reduce the information throughput of the network. The efficiency, **S**, of the token-bus network with transmitting nodes is thus:

$$S = \frac{T_d}{\{T_d + \{T_r/Q + T_f\}\}} \quad \ldots (6)$$

Where T_d is the average time taken to send the data.

T_r is the token rotation time defined by (5)

Q is the number of nodes.

T_f is the time spent sending framing bits.

The values of T_d and T_f are adjusted where multiple frames may be transmitted per token hold.

The two major causes of inefficiency are the number of nodes active on the logical ring that are not transmitting data frames, and the end-to-end propagation time. As the number of frame level overhead bits are fixed by IEEE, the variables are the number of non-transmitting stations active on the logical ring, the length of the network which effects the propagation time, the speed of transmission and the packet size.

Equation (6) was used to calculate the maximum throughput for this network, using the following parameter values:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = 256 octets.
5. Frame overhead = 108 bits.
6. Token frame = 9 octets.

These values allow a maximum of 441.18 frames per second thus giving a maximum throughput efficiency of 90.35%.

A smaller frame length means that a greater percentage

of the time is being used to transmit frame overhead as opposed to user information, thus reducing the throughput efficiency. Mathematically this means that the value of T_d in equation (6) is reduced which, in turn, affects the ratio of control to user data and thus the efficiency. Figure 2.9 below illustrates the effect of frame length on the throughput for the Token-bus network.

The IEEE 802.5 Token-ring Throughput:

In this architecture it is, again, the rotation of the token that is the major cause of inefficiency. However, in this case the network propagation time includes the NIU latency, i.e. the delay imposed by the NIU between receiving and relaying a bit. Because the IEEE 802.5 Token-ring is a single token network, which means that a station must receive its own start-of-frame marker before issuing a new free token, the maximum throughput is dependent on whether the frame transmission time is longer or shorter than the network propagation time.

It is clear that if the frame transmission time is shorter than the propagation time, the station will have to transmit idles until the start-of-frame marker returns, thereby reducing the throughput. In this case the maximum throughput efficiency function is given by:

$$S = \frac{T_d}{\{T_r + \{T_r/Q + T_f\}\}} \quad \ldots (7)$$

Where T_d is the average time taken to send the data.

T_r is the token rotation time defined by (8)

Q is the number of nodes.

T_f is the time spent sending framing bits.

$$T_r = T_p + Q \bullet T_l \quad \ldots (8)$$

Where T_p is the propagation time for the whole ring (excluding NIU latency).

Q is the total number stations on the ring.

T_l is the NIU latency.

For the case where the frame transmission time is longer than the network rotation time, (7) is redefined as follows because the limiting factor is now the transmission time of the frame and not the rotation time:

$$S = \frac{T_d}{\{T_d + \{T_r/Q + T_f\}\}} \quad \ldots (9)$$

Where T_d is the average time taken to send the data.

T_r is the token rotation time defined by (8)

Q is the number of actively transmitting nodes

T_f is the time spent sending framing bits.

The maximum throughput may now be calculated from equation (9) using the following parameter values:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = 256 octets.
5. Frame overhead = 104 bits.
6. NIU latency = 1 bit time.

Since the frame is 256 octets long, which equals a transmission time of 2.152 msec (for the data plus framing bits), while the network rotation time is only 31.667 usec (for 29 stations plus 800 m of cable), the maximum

throughput calculations used equation (9) instead of (7). The maximum throughput efficiency is therefore 95.12%.

The major contributers to the inefficiency are the latency of the NIU, the number of nodes that are not transmitting data and the packet size. The packet size determines the number of frames required, and thus the number of framing bits, for the supplied user data.

The figure 2.9 below shows the dependence of the maximum throughput on the packet size of the frame for all the above networks.

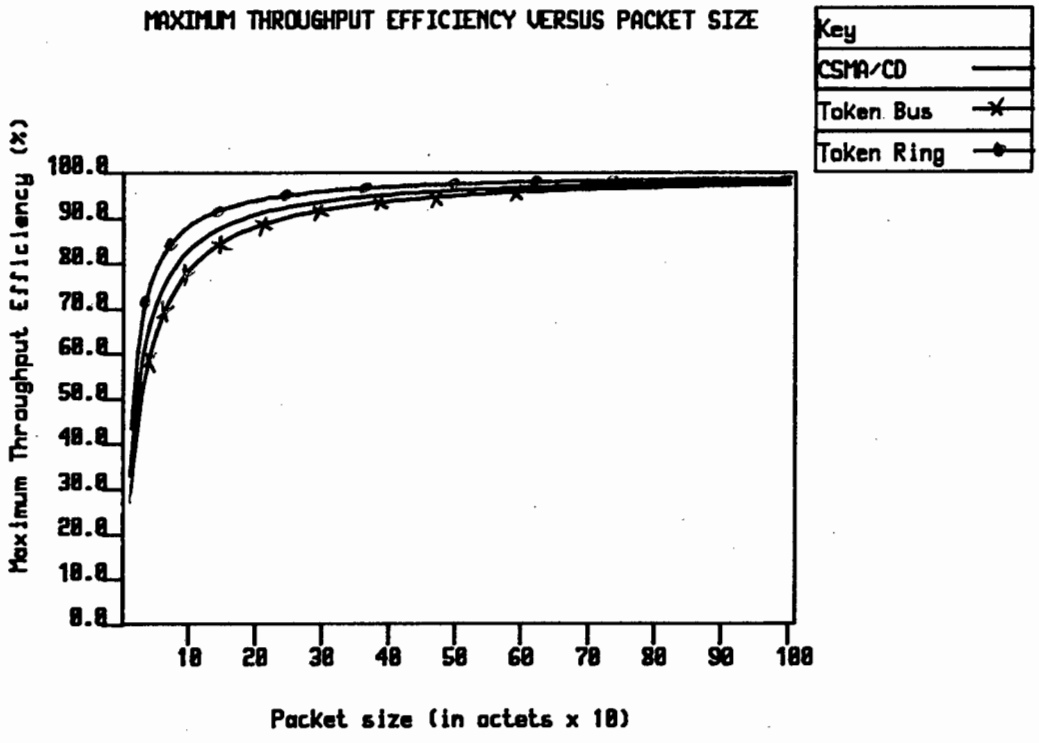


Figure 2.9 - Maximum Throughput versus Packet Size

As was expected, the maximum throughput improves for a larger packet size as the proportion of framing overhead to information bits is lower. Stallings [28] shows this trend for a larger range of packet transmission times.

Figure 2.9 also shows that there is little difference in this case between the maximum throughput for the three networks for a particular packet size and that the greatest effect is at low packet sizes. It is the Token-bus network that displays the lowest throughput due to the extra control information required because the token is regenerated at each node.

(b) Delay

Throughput is related to the ratio of information to the sum of the control and overhead bits on the network. The delay, on the other hand, might be defined as the amount of time taken by all seven network layers to transmit a unit of data between two application tasks. Included in this delay is the time taken to tranfer the data from the application task to the physical drivers, the time taken for it to be transmitted across the network, and the time required to pass it to the remote application task. In this section only the delay of the access control section of the NIU is considered. The queueing delay may thus be defined

as the time between data being presented at the LLC/MAC layer interface of the local computer to its presentation to the LLC layer by the MAC layer at the remote computer.

This delay is determined by, firstly, the rate at which the data is supplied to the MAC layer, and secondly, the capacity of the network.

Firstly, with regard to the rate at which the data is supplied, there is a cut-off point, at the maximum throughput rate of the network, after which the network imposes infinite delay on the supplied data. In practice the delay increases slowly at first and then approaches infinity as the maximum throughput is neared. This means that the load induced delay of a unit of data is determined by the proximity of the presented load to the maximum throughput of the network. (The maximum throughput is that value calculated for the different access methods in the previous section.)

The second part of the delay is due to the time taken by the network to carry the data frame from the source to the destination, i.e. the transmission delay. This is simply the length of the frame divided by the capacity of the network plus the propagation time for the network. The longer the frame, the longer the transmission delay as the

two are linearly related.

Bux, in reference [19], gives the transfer delay functions for the Token-ring, Slotted-ring, CSMA/CD and an ordered-access bus using the Multilevel Multiple Access (MLMA) scheme. The results for the CSMA/CD and Token-ring are given below, as well as the transfer delay function for Token-bus network, which was derived from the Token-ring delay function.

The CSMA/CD Transfer Delay Function:

Equation (10) below is based on a derivation by Lam [20]. This equation assumes that the CSMA/CD network does not become unstable and so produce infinite message delays under collision conditions but uses an adaptive algorithm, such as the truncated binary exponential backoff method, to ensure network stability. The arrival times of the MAC layer packets are assumed to be exponentially distributed. The transfer delay (**D**) for the CSMA/CD network may thus be described as:

$$D = \frac{\lambda\{E[T_p^2] + \{4e+2\}\tau E[T_p] + 5\tau^2 + 4e\{2e-1\}\tau^2\}}{2\{1-\lambda\{E[T_p] + \tau + 2e\tau\}\}} + E[T_p]$$

$$+ 2\tau e - \frac{\{1-e^{-2\lambda\tau}\}\left\{\frac{2}{\lambda} + \frac{2\tau}{e} - 6\tau\right\}}{2\left\{F_p^*(\lambda)e^{-\lambda\tau/e} - 1 + e^{-2\lambda\tau}\right\}} + \frac{\tau}{2}$$

... (10)

Where λ is the packet arrival time.

$E[T_p]$ is the mean service time for the packet, both header and data.

τ is the propagation time for the network.

$F_p^*(\lambda)$ is the Laplace transform of the probability density function of the packet service time T_p.

This equation was used in figure 2.11 below to illustrate the dependence of delay on load for the CSMA/CD network with the following parameter values:

1. Number of active stations = 30.
2. Data path length = 800 m.
3. Data rate = 1 Mbps.
4. Packet size = fixed length of 256 octets.

The Token-ring Transfer Delay Function:

A Token-ring network may be modelled as shown in figure 2.10 where there are as many queues as there are active stations attached to the network. The network services the queues in a fixed rotation, with a switching delay of τ_i between each queue's servicing. This switching delay is the sum of the NIU latency and the propagation delay to the following station. The packets are queued at a rate of λ_i.

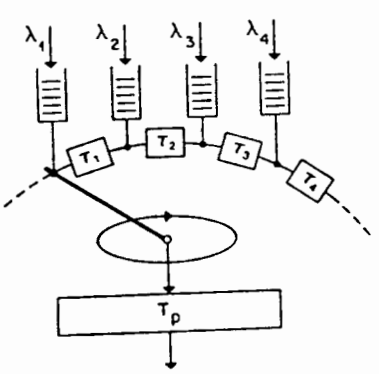


Figure 2.10 - The Token-ring Model

In the relationship derived by Bux [19], based on the discrete time approach model described by Konheim and Meister [26], both the packet arrival times, λ_i, and the switching times, τ_i, are equal. Equation (11) assumes that the discrete time interval approaches 0, and the arrival times follow the exponential distribution. The transfer delay is thus:

$$D = \frac{\rho E[T_p^2]}{2\{1-\rho\}E[T_p]} + E[T_p] + \frac{\tau\{1-\rho/Q\}}{2\{1-\rho\}} + \tau/2 \qquad \ldots (11)$$

Where λ is the packet arrival time.

ρ is the load:

$$\lambda E[T_p]$$

$E[T_p]$ is the mean service time for the packet, both header and data.

τ is the propagation time for the network.

Q is the number of active stations.

This model can account for single token network operation by setting the value of T_p to equal the ring rotation time, T_r, when T_p is less than T_r.

Equation (11) was used in figure 2.11 to describe the dependence of transfer delay on the load for a Token-ring network with the following parameter values:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = fixed at 256 octets.

5. NIU latency = 1 bit time.

The Token-bus Transfer Delay Function:

The Token-ring model, shown in figure 2.10, may also be used to describe the Token-bus. The stations on the bus are placed in an order and serviced in a fixed rotation, similar to the Token-ring. In the Token-bus the switching delay, τ_i , refers to the transmission of the entire token plus its propagation time to the next station, whereas τ_i for the token-ring refers only to the NIU latency (usually only one bit time long) plus the propagation time to the next station. Another exception is that the token is transmitted immediately after the completion of the last (allowed) frame in the Token-bus, so the value of T_p is not adjusted for frames whose transmission time is less than their propagation time.

The transfer delay is thus described by equation (11), except that the value of τ is the sum of the token transmission time and the propagation time. This equation was also used to describe the function of load versus delay for the Token-bus network with:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.

4. Frame length = fixed at 256 octets.

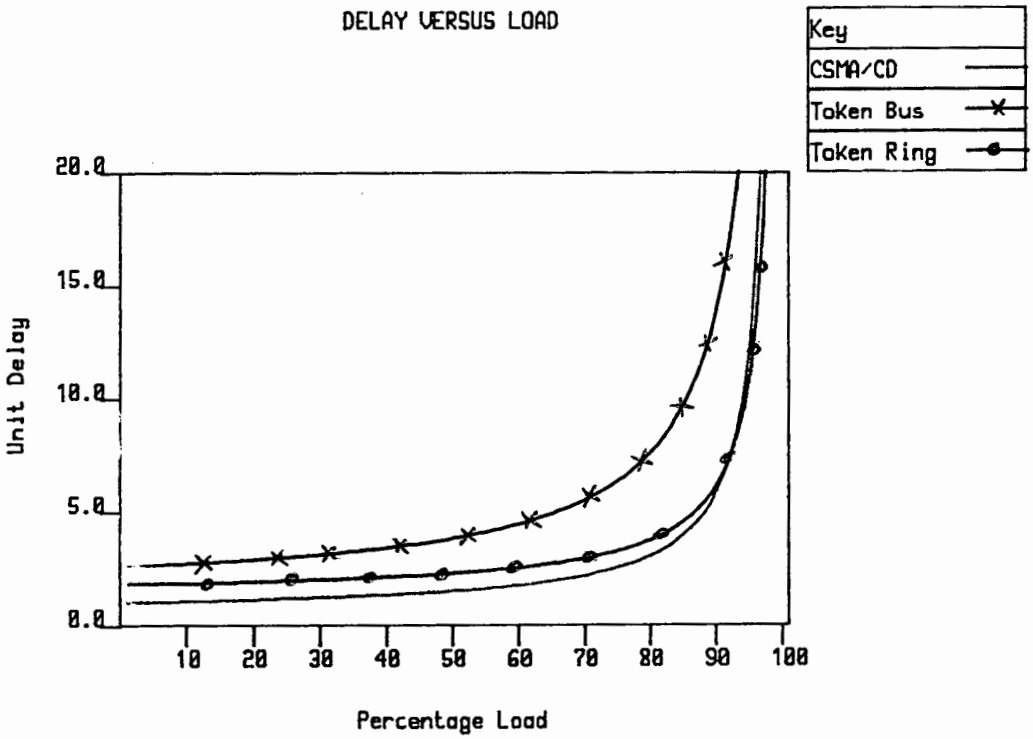


Figure 2.11 - Delay versus Load Characteristics.

The Token-bus network displays the greatest delay for these specific parameter values, irrespective of load. This is not generally the case as can be seen when studying the characteristics given by Acampora and Hluchyj [29]. They show that for a higher propagation delay to transmission time ratio (a) the CSMA/CD would perform significantly worse, for loads above 50%, whereas the Token-ring's delay would not increase to the same extent. The Token-bus will always give the greatest delay at low loads because of the long token rotation time, but for higher loads its delay is less than CSMA/CD's for large values of a.

Conclusions with regard to delay by Bux [19] and Stuck [21] are more comprehensive:

1. The transmission rate influences the delay on the CSMA/CD network more than the delay on the other networks. For instance, in the CSMA/CD network, the delay tends to infinity close to a load of unity for a 1 Mbps network, whilst on a 10 Mbps network infinite delay would be experienced at a load of about 65%.

2. A change in packet length distribution, from a constant distribution to one that had large variations (according to a hyperexponential formula), indicated that the delay induced by three networks was largely unaffected by the packet length distribution.

3. A greater NIU latency increases the delay of the token-ring as the frame rotation is increased. This can also be seen by examining equation (11) in which the propagation delay, τ, is effected by the NIU latency and so effects the transfer delay. This change in transfer delay is multiplied by the number of active stations.

4. A change in network length affects the propagation time and so the delay times for all three networks. Changing the network length whilst keeping the other parameters fixed has the inverse effect of adjusting the transmission rate. Thus, as in 1 above, CSMA/CD is affected most by an increase in network length because the vulnerable period of the transmission, again, is increased. The two token networks are affected only in as much as the propagation delay affects the token rotation and frame transmission times.

An important conclusion reached by Murray and Enslow [25] and Stuck [21] is that the delays caused by the higher level protocol and the interface circuitry are the major time consumers and that, by comparison, the access method does not significantly influence the time taken for data transmission. The efficiency of the control software is therefore more important than the access method used.

2.3.2 Reliability

Two factors which often influence the choice of a network architecture are its resilience under error conditions and its ability to recover quickly from their

occurrence. Phinney and Jalatis [23] propose a number of error sources in their discussion of the error handling capabilities of the IEEE 802.4 bus. These are used here to examine the error handling abilities of the proposed architectures.

The error sources are:

1. Signalling and Noise Errors
2. Medium Failure
3. Transmitter and Receiver Failure
4. Repeater Failure
5. Station Malfunction
6. Station Failure

Signalling and noise errors are the result of environmental conditions such as impulse (terrestrial) noise or Rayleigh (shot) noise. These errors may also be a result of non-linear medium characteristics. Their occurrence would cause infrequent and intermittent errors on the network.

The medium may fail because of shorts or open circuits caused by, for example, moisture, bad connections or breakage.

Transmitter failure may manifest itself in two ways.

Firstly, the transmitter could continuously send data (jabber) or, secondly, the data rate or transmission power could drift. Receiver failure could manifest itself as deafness or signal attenuation. The frequency of occurrence of these errors is dependent on the design of the station. For instance, a jabber control is generally required so that a jabbering station does not cause total network failure for a prolonged period.

If a repeater node fails it may divide the network in two, allowing it to continue but as two seperate networks. If the failure is in one direction only, part of the network will continue as a seperate network whilst the other part will fail because it will hear network activity but will not be able to take part.

A station malfunction is due to some failure on the MAC layer, e.g. the station may detect errors in the frame check sequence or in the address when, in fact, they are correct.

Station failure could be caused by power failure or a higher-level command excluding it from the network.

CSMA/CD Error Handling:

The CSMA/CD network detects error conditions via the analogue collision detection hardware and the 32-bit frame check sequence (FCS).

It is unable to distinguish between short noise bursts and frame collisions, clearly a disadvantage, so there is no way of detecting short errors which trigger the collision detection hardware as errors and not collisions. This form of error will be recovered from by the frame collision re-transmission process. Should the noise error be too short to be seen as a frame collision, the corrupted bits will be detected by the FCS appended to the frame. A recovery procedure will then be initiated, usually leading to a frame re-transmission.

Medium failure will cause network failure as data cannot be transmitted under such conditions. Where jabbering occurs (case 3), the whole network will be disabled as the network will always be seen to be active, thus disallowing access to other stations. If the station fails in any other way (cases 5 and 6), only the offending station will be disabled.

Error Handling on a Token-bus:

Bit errors are detected by means of a 32-bit Frame Check Sequence (FCS).

An error of type 1, i.e. noise errors, will corrupt one or more bits in any frame. The FCS will detect the error and recovery is achieved by retransmitting the corrupted frame.

Should the medium fail, the whole network will be brought to a standstill as no data can be transmitted.

Where the transmitter malfunctions and begins to jabber, the network will fail until the jabber timer halts the failed station. In the other causes of station failure or malfunction, detailed above, the responsible station will be excluded from the logical ring at the next token pass. If the station fails while holding the token, the lost token error recovery procedure, given in the first part of this chapter, will be invoked.

The failure of a repeater will result in the isolation of a part of the network or the failure of the entire network, depending on the severity of the failure.

Error Handling on the Token-Ring:

Both Bux et al. [22] and Saltzer et al. [24] discuss the error recovery features of the Token-ring. In the Token-ring network any of the stations may be given the task of "monitor". It is the responsibility of the monitor station to maintain the network in a useable state and so it has the task of initialising most of the recovery procedures.

Signalling and noise errors can cause the loss of single frames, including the token. If the token is lost, the monitor's rotation timer will expire. It will then clear the network by transmitting a string of idle characters. Once they have been transmitted around the network and are received by the monitor, it will issue a free token and the network will continue as before. The corruption of any other frame will be detected by the FCS and a re-transmission will be requested by the destination node.

If either the link between any two nodes or one of the nodes fails, the network will be disabled unless some precaution is taken to isolate the disabled section. This is generally accomplished with the aid of isolation relay switches. The act of isolating the failed station or link

will result in the corruption of at least one frame, and perhaps the loss of a token. Recovery is as described above.

Two other conditions must also be considered in a Token-ring: a continuously circulating busy token and a duplicate token condition. The first condition could be caused by noise adjusting the free-token bit in the header, thus converting a free token into a busy one. The duplicate token condition may be caused by noise converting a busy token into a free one, which will allow a second station access to the network while the first is still transmitting. (The IEEE Token-Ring is a single token network.)

A circulating busy token is detected by the Monitor bit in the access control field of the frame header. This bit is always cleared by the originating station when it transmits a frame, and is set when a frame passes the monitor station. Should it be set in a frame received by the monitor node, meaning that the originating station has not reset it, the monitor will clear the ring by means of idle characters and then initiate the transmission of a free token.

A duplicate token error will be discovered by the token-holding station as the station will either read an

incorrect source address in a received frame, resulting from another node having begun its transmitting simultaneously, or it will receive a second token. In both cases the node will complete its frame transmission but will not issue a free token. After a period the monitor will discover that there is no token and perform a lost token recovery procedure.

2.3.3 Physical Level Considerations

The third aspect that influences the choice of an architecture is the physical constraints of the network, e.g. excessive cabling requirements and poor noise immunity. A description of the important physical considerations of the networks under scrutiny follows.

Physical Aspects of the CSMA/CD Bus:

With the listen-while-talk feature of CSMA/CD, the collision detection section of the interface unit must be able to sense the presence of a distant transmitter while being in close proximity to the local transmitter. This is achieved by sensing analogue levels on the network and comparing them to those levels being placed on the network by the transmitter. Because the difference in these levels can be small, sensitive analogue circuitry is required.

This analogue component is clearly a constraint.

Physical Aspects of the Token-bus:

There are a variety of signalling methods available to the IEEE 802.4 network designer, e.g. phase-continuous FSK and duobinary AM/PSK. In each case co-axial cable is used, making it prone to noise, and care must be taken to ensure that earth loops do not exist.

The sensitive analogue interfaces, required by CSMA/CD, are not required as the access method is entirely digital.

Physical Aspects of the Token-ring:

If a node or link on the Token-ring fails, the whole network will be brought to a standstill. Saltzer et al. [24] suggests that each link should be looped through a central hub. A relay, situated in the hub, can be activated remotely by the station. If either a link or the station itself should fail, the relay will bypass the station by shorting the failed station's input and output at the hub.

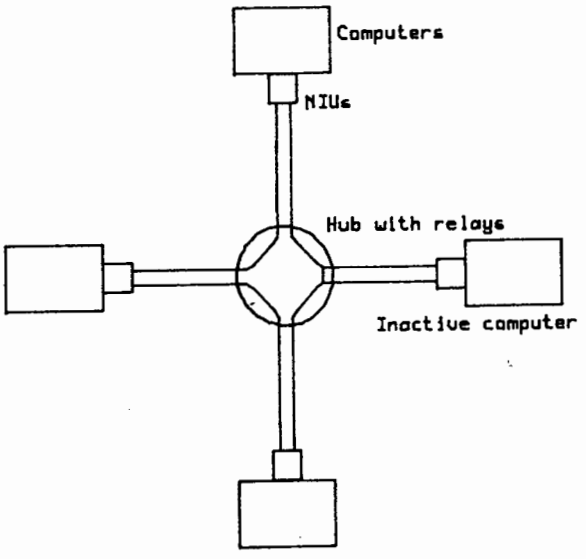


Figure 2.12 - Token-ring Hub

Ring synchronization: The ring is a closed loop and if there is no synchronization around the ring, data will be interpreted incorrectly. Clock recovery, and therefore synchronization, is achieved by synchronizing a local oscillator to the received data by detecting zero crossings. Owing to noise or imperfections in the clock recovery circuit, these zero crossings are distorted and cause deviations from the mean frequency in a random fashion, called "jitter". As the clock recovered from the received data is used to transmit the frame, the jitter can accumulate around the ring. In order to correctly track the data from the previous node, a narrow band Phase-Locked Loop (PLL) is required at each node. To maintain stability, the network should be broken at the monitor node to supress the

accumulated jitter. The data is then received by means of a PLL and written to an elastic first-in-first-out (FIFO) buffer from where it is transmitted at a frequency fixed by a crystal oscillator.

2.4 Conclusions

This chapter has compared the IEEE 802 networks with reference to their performance, their reliability and their physical constraints. Of the IEEE 802 networks the Token-bus was chosen because:

1. It had the least complex physical level.

2. The difference in throughput and delay between the Token-bus and the other networks, for the specific requirements of this network, was not significant.

3. The Token-bus is at least as reliable as the other networks.

The introduction of the WD2840 provided two more reasons for choosing the token-bus, viz.:

1. It implements a version of the complex MAC layer,

thereby permitting a quick and simple network implementation and is also inexpensive and easy to use.

2. Single chip controllers for the CSMA/CD and Token-ring networks were not available.

References for Appendix J

27. WITTLIN, R.I., RATNER, D.V., "Choosing the Best Local Area Network for an Application", Computer Design, pp 143-149, Feb 1985

28. STALLINGS, W., "Local Network Performance", IEEE Communiactions Magasine, Vol. 22, No. 2, pp 27-36, Feb 1984

29. ACAMPORA, A.S., HLUCHYJ, M.G., "A New Local Area Network Architecture using a Centralised Bus", IEEE Communications Magasine, Vol. 22, No. 8, pp 12-21, Aug 1984